WL-TR-91-1146

STRETCH AND HAMMER NEURAL NETWORKS FOR N-DIMENSIONAL DATA GENERALIZATION

Peter G. Raeth
Electronic Warfare Support Measures Research Group
Avionics Directorate (WL/AAWP-1)
USAF Wright Laboratory
Wright-Patterson AFB, Ohio 45463-6543

Steven C. Gustafson, Gordon R. Little, Todd S. Puterbaugh
University of Dayton Research Institute
Kettering Laboratory; KL-463
300 College Park
Dayton, Ohio 45469-0140

15 January 1992

Final Report For Period May 1991 - October 1991

Approved for Public Release; Distribution Is Unlimited.

Approved for Public Release; Distribution Is Unlimited.

Avionics Directorate
Wright Laboratory
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433-6543

NOTICE

This technical report has been reviewed and is approved for publication.

[illegible]
PETER G. RAETH, MAJOR, USAF
Program Manager
ESM Research Group Avionics Directorate
Passive Electronic Countermeasures Branch

[illegible]
PAUL S. HADORN, PhD, Chief
Passive ECM Branch, EW Div.
Avionics Directorate

[illegible]
JOHN SEWARD, LT COL, USAF
Chief, Electronic Warfare Division
Avionics Directorate
Wright Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 1992 January	**3. REPORT TYPE AND DATES COVERED** Final Report, May 91 to Oct 91

4. TITLE AND SUBTITLE
STRETCH AND HAMMER NEURAL NETWORKS FOR N-DIMENSIONAL DATA GENERALIZATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Peter G. Raeth, Steven C. Gustafson, Gordon R. Little, Todd S. Puterbaugh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Avionics Directorate
University of Dayton Research Institute

8. PERFORMING ORGANIZATION REPORT NUMBER
WL-TR-91-1146

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Avionics Directorate, Wright Laboratory, Wright-Patterson AFB OH 45433-6543

University of Dayton Research Institute, 300 College Park, Dayton OH 45469-0140

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
A hyper-surface stretch and hammer neural network has been developed that generalizes data from processes that have one output variable and one or more input variables. This network achieves several desirable properties through a novel combination of standard methods. The methods incorporate principal components, linear least squares, Gaussian radial basis functions, and diagonnally dominant matrices. An easily visualized physical model of network function ensures that the combination of methods is appropriate and practical. The model has natural potential for parallel implementation and for n-dimensional classification and other pattern recognition tasks. These tasks include smoothing (interpolation), filtering, and prediction (extrapolation). The model can be extended to accommodate multiple outputs. Unlike many other neural networks (such as backpropagation-trained networks), the training and performance characteristics of the stretch and hammer neural network. The trials on three-dimensional surface interpolation are also presented, as are notes on other potential applications.

14. SUBJECT TERMS

15. NUMBER OF PAGES 101

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Acknowledgements

The authors are indebted to the following people:

Dr. John Loomis and Dr. Allan Lightman for discussions on computer aided drafting and computer aided manufacturing.

Dr. David Flannery, Mr. William Phillips, and Mr. Nicholas Pequignot for the use of computing hardware.

Ms. Wilma Barnes and Ms. Debbie Ables for administrative support.

Major Robert Landry for facilitating the fellowship during which this research took place.

The sponsors of this research:

University of Dayton Rapid Prototype Development Laboratory
University of Dayton Applied Physics Department
USAF Wright Laboratory Avionics Directorate
USAF Wright Laboratory Materials Directorate

Table of Contents

List of Figures

List of Figures (cont'd)

1. Problem Description

A critical problem in the link between computer-aided-drafting and computer-aided-manufacturing (CAD/CAM) is the volume and complexity of data that must be sent between CAD computers and rapid prototyping machines. The research project reported here was designed to apply neural networks to this problem. The basic question was, "Do neural networks enable a decrease in the grid sampling density for surface interpolation in solid modeling for CAD/CAM?"

Rapid prototype development (RPD) is a computer-aided-manufacturing technique for producing pre-production parts directly from the parts' CAD representation. The technique is also used for small batch runs and for producing molds. According to Kirshman et al., "Rapid prototyping technology is perhaps the most significant new concept in manufacturing since numerical control machine tools." According to Hull, there are seven critical areas in rapid prototyping:

1) Part size
2) Building speed
3) Building accuracy
4) Physical properties of formed parts
5) Ease of use
6) Reliability
7) Process benefits and costs in the overall manufacturing framework.

The research reported here was mainly concerned with building accuracy but building speed, ease of use, and reliability were also considered. It had been hoped that process benefits and costs also would be addressed, but an advantage over traditional interpolation schemes for this application was not shown (see Conclusions).

Reliability was addressed by using a neural network whose training and performance characteristics are predictable. The stretch and hammer neural network (SHNN) trains in a number of steps and uses an amount of resources that are predictable before training begins. The same is true of performance once training is completed. The amount of resources and the throughput time are known before the network is trained or implemented.

Ease of use was addressed in that the SHNN does not require the user to set any internal network parameters. The SHNN is fully self-adjusting to the problem at hand.

Not all neural networks have these features of reliability and ease of use.

Building speed is affected by the amount and complexity of data that must be transferred to the rapid prototyping machine from the CAD computer. The SHNN was investigated as a way to minimize the density of the sampling grid needed to represent the surface of a given part. An alternative that was considered was the use of the SHNN to enable the use of larger surface facets. In both cases, it was hoped that data of less volume and complexity would need to be transferred. Hull states that past improvements in this area have come from various data compression schemes and faster data communications. He maintains that data preparation is still the slowest portion of the CAD/RPD process for complex parts.

For building accuracy we investigated the amount of data (and thus the density of the sampling grid) needed to achieve CAD/CAM precision (0.001). The hope was that CAD/CAM precision could be achieved by the SHNN using a less dense sampling grid than that required by traditional interpolation schemes.

Donahue and Turner have also noted the large file sizes that have to be transferred for a given precision. They state that "... current information transfer methods coupled with the differences in CAD representation schemes provide ample opportunity for improvement in the CAD to rapid prototyping process ..." Heller notes that, "One of the largest hurdles to cross at this stage of rapid modeling is the data transfer nightmare." He cites three major solid modeling methods:

1) Polygonal: representation by a collection of triangular-shaped facets
2) Constructive solids: representation using standard shapes as building blocks
3) Surfacing: representation by splines or polynomials

The effort reported here focused on the surfacing method and used Gaussian radial basis functions instead of splines or polynomials.

2. Brief Overview of Neural Networks

From an applications point of view, the field of neural networks investigates ways to program massive arrays of parallel processors to perform useful tasks. The interest in this field stems from the difficulty of using traditional sequential software methods to meet modern requirements. Another impetus to the field is that the traditional uni-processor architectures are increasingly the cause of bottlenecks that prevent timely task completion, especially in real-time environments. Finally, neural networks can generalize from examples composed of multiple data elements.

Neural networks are massive arrays of simple processors that execute in parallel. These processors are typically arranged in layers (see Figure 2-1). The processors in one layer are usually fully connected with the processors in the immediate neighboring layers. A processor is sometimes connected to itself and to other processors in its own layer. The connections between processors have associated weights that modify data flowing through the connections. Each processor executes (in parallel with the other processors in its own layer) a weighted summation or product of its input data elements, an intermediate non-linear transfer function, and an output function. The "program" of a neural network is contained in the inter-processor connection weights. There can be hundreds of thousands of these connection weights.

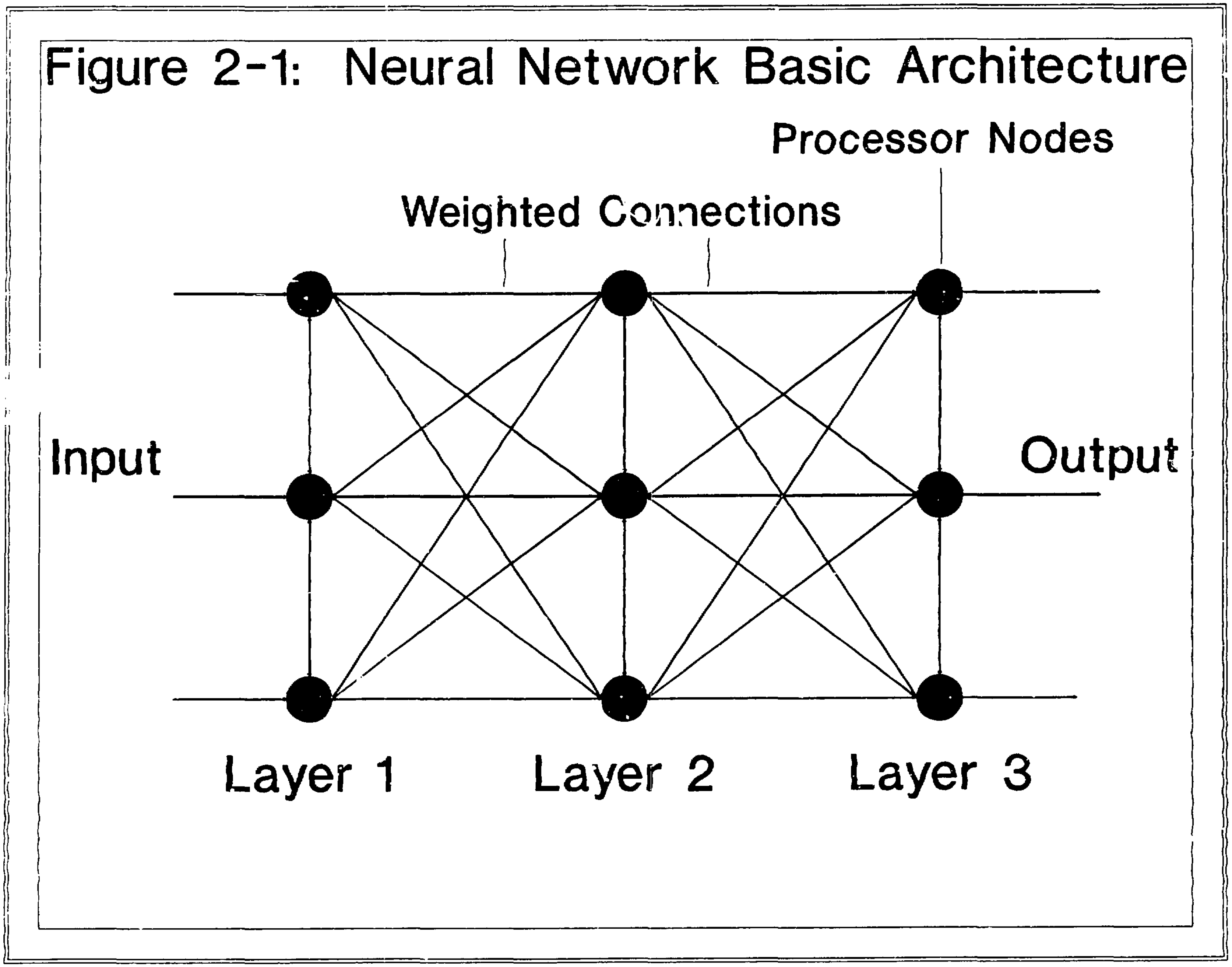

Figure 2-1. Neural Network Basic Architecture

There are many methods for setting the connection weights (sometimes called training the network). Not all methods are logistically supportable (See Raeth. Logistical supportability refers to modifiability and maintainability in the face of changing requirements). All methods involve sending "training" data through the network. These data represent examples of the task the network is to accomplish. As the training data pass through the network, the weights are adjusted automatically according to one of several training methods. The network will respond appropriately to the training data given that the network has had a sufficient exposure to the training data. If the training data adequately represent the task to be accomplished, the network also can correctly process test data that it has not been trained on.

Neural networks are not programmed in the traditional sense. Rather, they adjust themselves to the task at hand based on examples. Computers programmed via traditional sequential methods learn from algorithms composed of explicit task-accomplishment instructions. Neural networks are heuristic in nature, not algorithmic. Because of this, training a neural network is not as straightforward as it might first appear. There are many training methods and many network architectures. Depending on the task at hand, a given training method and network architecture may or may not be appropriate.

The training and performance reliability of a neural network is of primary concern for logistical and mission support reasons. Thus, it is necessary to use a neural network architecture and training method that has predictable training and performance characteristics. Such a network is the stretch and hammer neural network discussed in this report.

Klimasauskas et al., Lippman, Rogers et al., and Rumelhart & McClelland have all written more detailed introductions to neural networks. Lippman's paper is easy to follow and is widely referenced. The other authors have produced full-length books. Klimasauskas and Rumelhart & McClelland provide IBM-PC disks with example networks. For increasing length and level of detail, start with Lippmann then go on to Rogers. Follow up with Klimasauskas. Rumelhart & McClelland is the most theoretical of the four.

3. Introduction to Stretch and Hammer Neural Networks

Stretch and hammer neural networks are members of the more general class of radial basis function networks. The Probabilistic Neural Network defined by Specht and further described by Maloney is also a member of this class as are the networks described by Zahirniak.

For three-dimensional solid modeling, stretch and hammer is best understood as a surface fitting or surface interpolation neural network. In this context the network has two phases: training and operation. In any supervised neural network, the training phase uses example inputs and expected outputs to adjust the weighted connections. When training is completed whenever the example inputs are presented, the expected output is produced. In solid modeling, selected (x,y) coordinates are used as example inputs and the height of the surface above some table (or baseline) is used as the expected output. In testing or operation, inputs that were not used for training are provided and an output is produced. For solid modeling, the neural network is expected to deliver as output a very accurate height for all (x,y) coordinates of the surface in question.

Briefly stated, the training of a stretch and hammer neural network is described as follows. (More details are provided in the paper by Gustafson et al. and in later sections of this report.) Orthogonal coordinates with two horizontal input axes and one vertical output axis are established. The training points can then be plotted in the coordinates of the resulting three-dimensional space. These points are stretched so that they are evenly distributed in the input (horizontal) space. A malleable plane is positioned to minimize the sum of squared vertical distances between the plane and the training outputs. The malleable plane is hammered at each training output by directing the hammer along each vertical least-squares line with normally and radially distributed accuracy using many small strikes. The variances of the resulting Gaussian radial functions are set so that the number of strikes at any training point just exceeds the number of strikes at all other training points. The hammering is stopped when the malleable plane is deformed to intersect each training output.

Testing is conducted by projecting the test inputs vertically from the horizontal plane to the vertical surface generated during training. The output is the vertical height of the surface from the horizontal plane at the (x,y) coordinates of a given test input.

Poggio and Girosi have also interpreted neural network learning in terms of hyper-surface construction. Such an interpretation also can be given to Specht's development of the Probabilistic Neural Network (PNN) although the surface developed by the PNN places radial Gaussian functions that have one of only two different heights at each training point. Thus, the PNN is less general than the stretch and hammer neural network and is useful for classification but not for continuous interpolation. The SHNN can be used for both tasks.

4. Stretch for Data Preparation

Aside from selecting appropriate inputs, preparing the input data for use in training and testing is perhaps the most critical process in neural network operation. Two data preparation methods are in common use: normalization and transformation by principal components.

Normalization takes the vector of input values and scales the elements so that they are bounded to a range of values. Normalization also can be used to ensure that the sum of input elements is bounded to a fixed value or to a geometric surface. The constraints imposed by many types of neural networks require normalization of some kind to be performed on the data inputs, network outputs, or on inter-layer node outputs. Normalization is not required by the SHNN and so it is not discussed further in this report. Normalization is typically discussed in the literature relative to specific types of neural networks.

Principal component analysis is a well-known statistical technique that is useful as an input data preparation step for the SHNN. According to Kruskal, principal component analysis allows reduction or elimination of indeterminacy. Translational indeterminancy is reduced by adding various constraints, such as constraints that force the data element mean to zero. Rotational indeterminacy is reduced by rotating the input vectors to principal coordinates. Principal coordinate axes form an orthogonal system in which the input data vectors are uncorrelated. Dilation indeterminancy due to relative scaling of the data elements is reduced by forcing the sum of the norms of the data elements to unity.

Principal component analysis also can be used to eliminate input data elements that have little variation relative to other elements. This function is accomplished by choosing for elimination those input data elements associated with the smallest magnitude eigenvalues of the data covariance matrix. Thus, the dimensionality of a problem may be reduced. (This procedure is described further by Hecht-Nielsen and Hertz, et al.) In the SHNN, however, the eigenvalues are used to "stretch" the small-variation elements in the principal component space so that the maximum variation is achieved. Thus, no information is lost and maximum use of all available information is achieved.

A more complete treatment of principal component analysis is given by Hotelling. The specific implementation of the principal component transformation used in the SHNN transforms all input data vectors based on an analysis of the training inputs. This "stretching" transformation finds linear combinations of the training input elements that are optimum in that the transformed-coordinate covariance matrix is the identity matrix.

Let x_{ij} be training element j for training vector i, where there are n elements j = 1,2, ..., n and m vectors i = 1,2, ..., m. Here, n is strictly less than m and the elements are assumed to be real.

The mean inputs are

$$\overline{x_j} = \frac{x_{1j}+x_{2j}+ \ . \ . \ . \ +x_{mj}}{m}$$

The inputs relative to the means are

$$x'_{ij} = x_{ij} - \overline{x_j}$$

Let **X'** be the matrix of $\{x'_{ij}\}$ of inputs relative to the means. The corresponding covariance matrix **A** is (T refers to the matrix transpose)

$$A = \frac{X'^{T} X'}{m-1}$$

The orthogonal eigenvectors $\mathbf{v}_j$ and the corresponding real eigenvalues λ_j associated with matrix **A** are the solutions of

$$Av_j = \lambda_j v_j$$

where: $\mathbf{v}_j$ is column j of the matrix of eigenvectors $\{v_{qj}\}$
q = 1,2, ...,n, and the eigenvalues are solutions of

$$|A-\lambda I| = 0$$

Let u_{ij} be input j in principal component coordinates for training vector i. Let $\mathbf{u}_i$ be the column vector formed from row i of the matrix of $\{u_{ij}\}$. Let $\mathbf{x'}_i$ be the column vector formed from row i of the matrix $\{X'_{ij}\}$. Then the principal component transformation is

$$u_i = Bx'_i$$

where the elements of the transformation matrix **B** are

$$b_{qj} = \frac{v_{jq}}{|v_q|\sqrt{\lambda_q}}$$

The numerical evaluation of the eigenvectors and eigenvalues is best accomplished using singular value decomposition of the matrix **A**. Note that the transformed inputs u_{ij} are unit-less and that the columns of the matrix $\{u_{ij}\}$ have zero mean, unit variance, and zero covariance. Singular value decomposition is a mathematical technique for dealing with systems of equations that are singular or numerically very close to being singular. Our implementation follows Press, et al., and is supplemented by Nobel and Daniel.

A simple two-dimensional example that employs the above algorithm is as follows. Choose the original coordinate values in the training space as shown below and plotted in Figure 4-1.

x-coordinate	Expected SHNN Output
5.0	1.0
2.0	2.0
3.0	3.0
1.0	4.0

Note that the SHNN does not require any particular order in the training data. The above x coordinates are transformed to the coordinates plotted in Figure 4-2.

x-coordinate	Expected SHNN Output
1.3175	1.0
-0.4392	2.0
0.1464	3.0
-1.0247	4.0

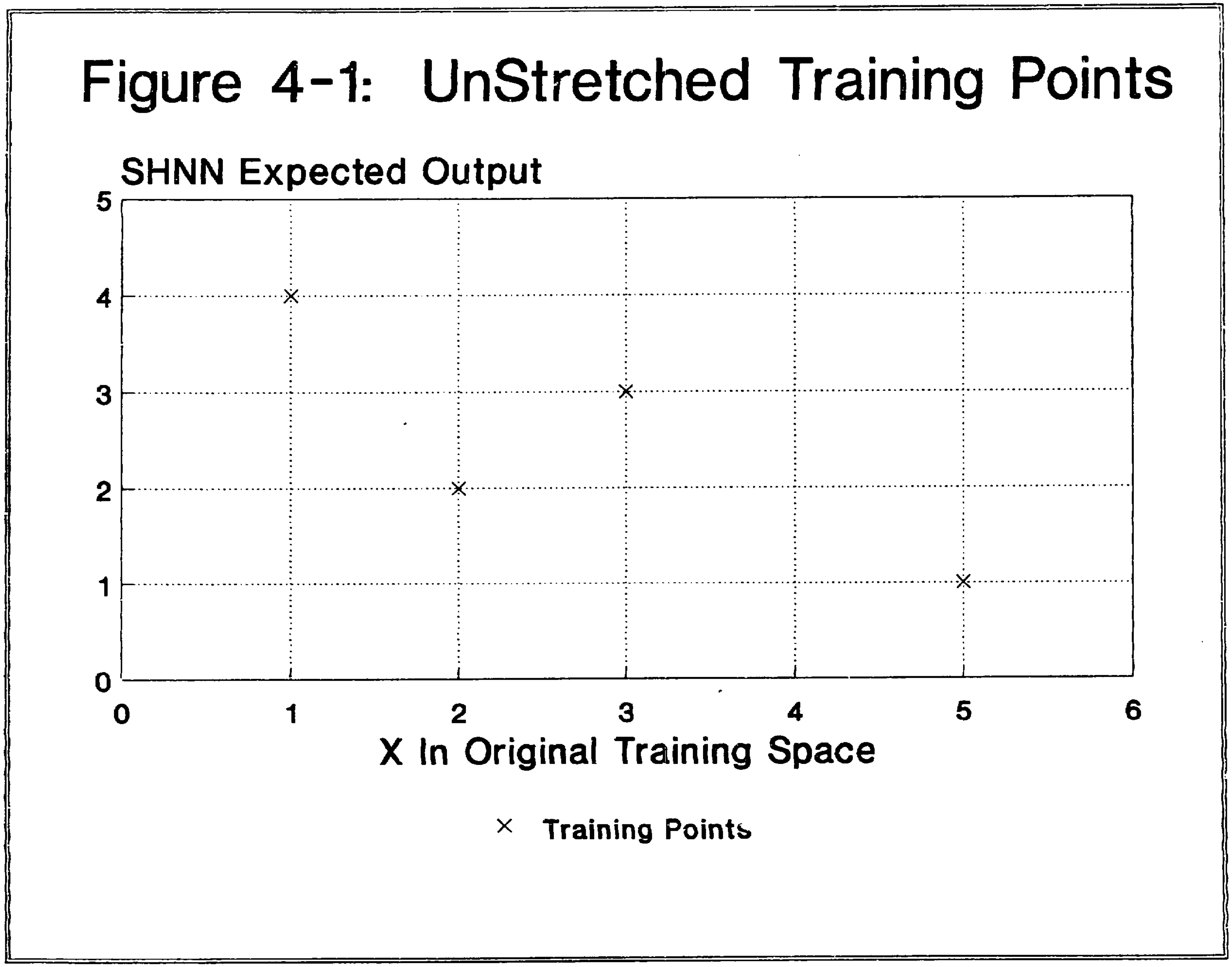

Figure 4-1. UnStretched Training Points

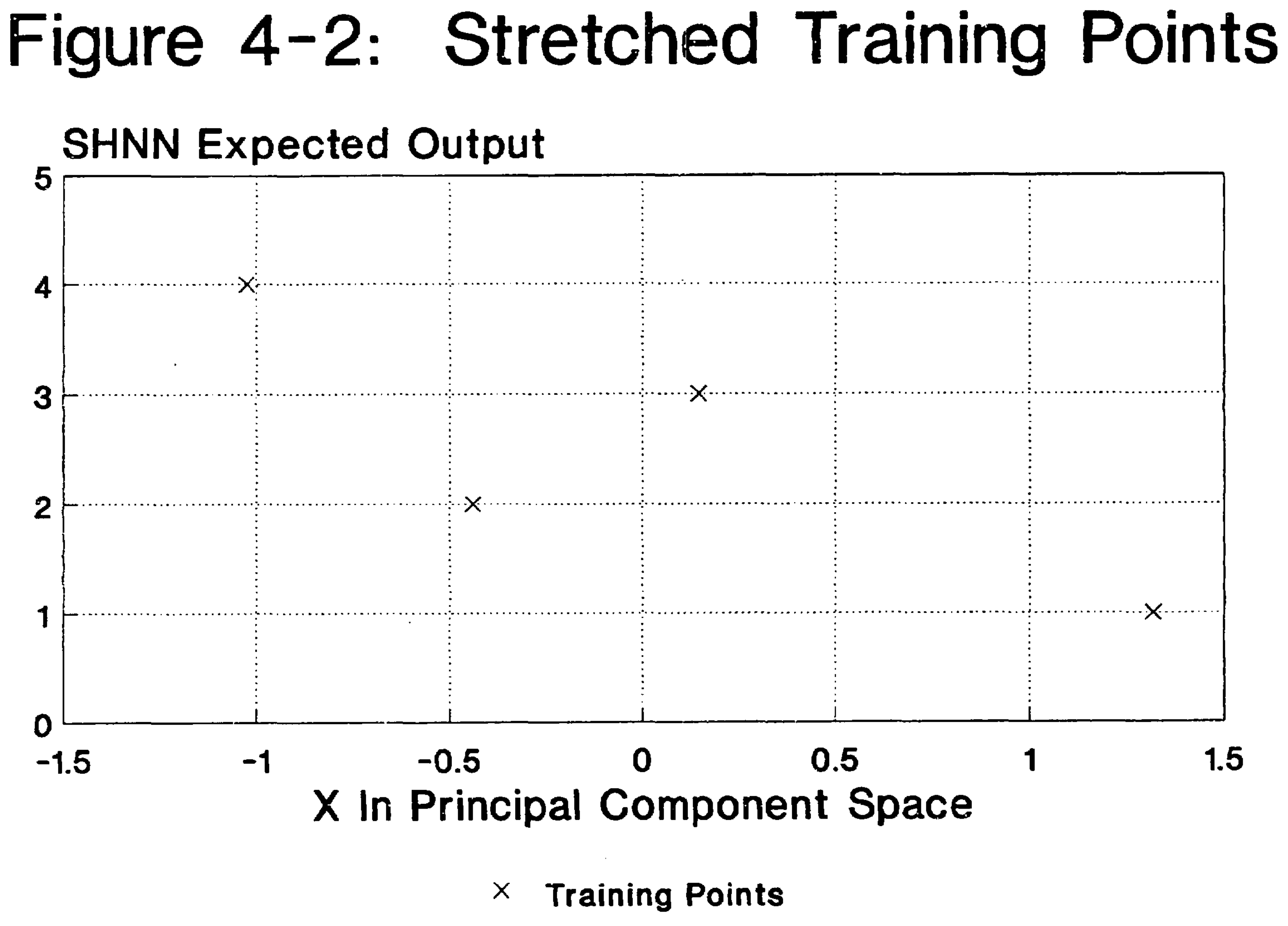

Figure 4-2. Stretched Training Points

Note that the Expected SHNN Output is not affected by the coordinate transformation and that the transformed coordinates are in principal component space, not the original training space. As desired, the average value of the transformed element values is 0.0 and the covariance matrix is the identity matrix, which in this case has a single unit element. The Expected SHNN Output is defined by the network user as the value to be produced by SHNN when the given coordinates are applied as input.

5. Hammer for Training

There are three phases to the training of a stretch and hammer neural network. Phase 1 fits a least squares plane to the training data. Phase 2 places Gaussian radial basis functions at each training point. Phase 3 uses those Gaussian functions to "hammer" the least squares plane until it contacts the expected network output at each training point.

Phase 1 of SHNN training fits or positions for "hammering" a linear hyper-plane of n dimensions to the training outputs using a least squares criterion. Depending on specific requirements, the training data is first be prepared using the principal components transformation (see Stretch for Data Preparation). Let **z** be the vector of training outputs $(z_1, z_2, ..., z_m)^T$, where T refers to the vector transpose. Let vector **a**, $(a_0, a_1, a_2, ..., a_n)^T$, be the coefficients of the linear hyper-plane, where n is the number of elements, a_0 is the z-axis intercept, and $a_1, a_2, ..., a_n$ are multipliers for the corresponding training vector elements. Let C be the matrix for which row i is $(1, u_{i1}, u_{i2}, ..., u_{in})$, where $\mathbf{u}_i$ refers to one of the m training vectors. Then the least squares solution of

$$z = Ca$$

fits the linear hyper-plane to the training outputs. Note that the numerical evaluation of the unknown elements in vector **a** is best accomplished using singular value decomposition and that the solution involves m linear equations in n+1 unknowns, where m is strictly greater than n. Continuing the example from the section, "Stretch for Data Preparation," Figure 5-1 shows the least squares line that fits stretched training points. The vector, **l**, of points on the least squares line are calculated using the following equation after **a** has been resolved:

$$l = Ca$$

where vector **a** = $\begin{matrix} 2.500 \\ -1.073 \end{matrix}$

Phase 2 of SHNN training places Gaussian radial basis functions at each training point. The variances for these functions are adjusted so that the matrix of Gaussian equation results is diagonally dominant by columns. In traditional interpolation methods, lines, polynomials, or splines connect adjacent training points. The SHNN adds up a series of Gaussian curves, where each curve is centered on a single training point. The variance of any given curve is such that the number of hammering strikes at the training point on which the curve is centered equals the number of strikes at all other training points.

Let **F** be the matrix of Gaussian functions which have an output of unity at their respective training points and an output which decreases as the distance to other training points increases. It is these functions which ultimately, in Phase 3, "hammer" the least squares plane (which was fit to the training points in Phase 1) to contact the training points.

Each column of the **F** matrix represents a given training point. Each element of each column also represents a given training point. Thus, the **F** matrix is of size m-by-m. The value stored at each matrix element is the result of the Gaussian function centered at the

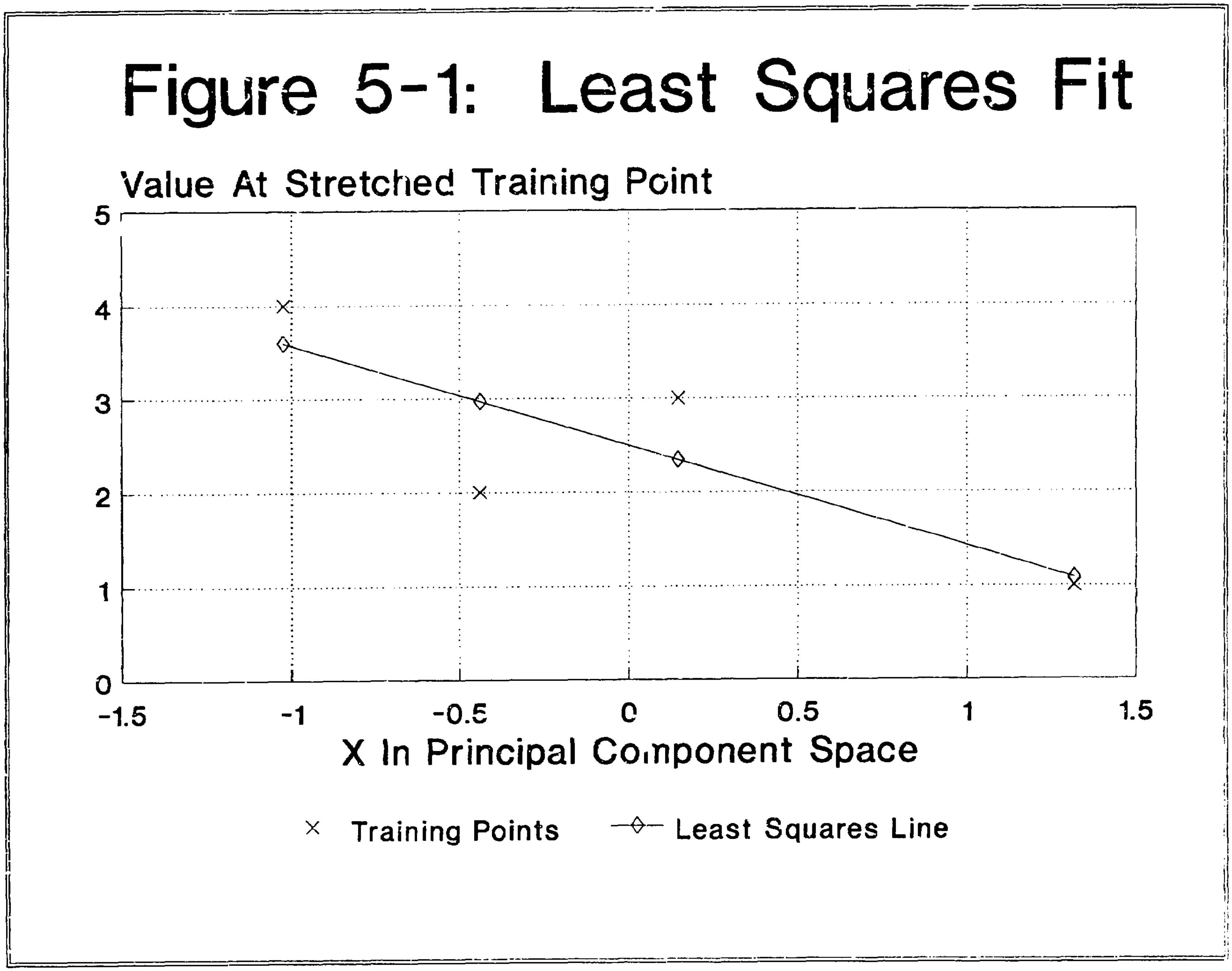

Figure 5-1. Least Squares Fit

column's training point as calculated at the training point indicated by the element. For example, column #1 indicates training point #1. The first element of column #1 indicates the first training point, the second element of column #1 indicates the second training point, and so on. Thus, column #1, element #2 stores the output for the first training point's Gaussian function calculated at the second training point. Note that the diagonal elements of matrix **F** are always equal unity and that the off-diagonal elements are always less than unity. The Gaussian equation employed to calculate the value of the $\mathbf{F}_{kj}$ matrix element is as follows:

$$e^{\frac{-\sum_{i=1}^{n}(u_{ji}-u_{ki})^2}{2\sigma_j^2}}$$

where:	u_{ij}	= the ith element of the jth training vector
	σ^2_j	= the Gaussian variance associated with the jth training point
	e	= the base of natural logarithms

For many choices of σ^2 the matrix **F** is singular. This results in a final network which does not fully represent the training data. A solution is to choose the Gaussian variances so as to ensure that the matrix **F** can never be singular. The approach which requires the fewest computations appears to be selecting the variances so that the matrix **F** is diagonally dominant by columns. Columnar diagonal dominance means that the sum of the absolute values of all off-diagonal elements in a given matrix column is less than the absolute value of the diagonal element in that column.

Physically, columnar diagonal dominance occurs when the variances are sufficiently narrow that the Gaussian function's value at neighboring (and, therefore, distant) points is small. Since extremely narrow variances would result in a pin cushion interpolating surface having poor smoothness, it is reasonable to attempt to make the variances as large as possible while maintaining diagonal dominance. This condition is achieved using a short iterative procedure. This procedure sets each column's variance to a fairly small value and then sums the off-diagonal elements in that column (in the case of Gaussians, the result is always positive). If the summation is not less than unity within some margin of error (say 0.001), then the variance is appropriately modified and the addition is repeated. This procedure is performed for each column in the **F** matrix. The **F** matrix and the variances for our continuing example are shown below.

The **F** matrix:	1.0000	0.0019	0.1760	0.0055
	0.2985	1.0000	0.6477	0.7220
	0.5843	0.4988	1.0000	0.2718
	0.1165	0.4988	0.1760	1.0000
Variances (σ^2):	1.2760	0.2965	0.3948	0.5264

Figure 5-2 shows the four Gaussian functions relative to the stretched training points and the least squares line.

Phase 3 of SHNN training uses the Gaussian functions developed in Phase 2 to "hammer" Phase 1's least squares plane so that it contacts the training points. The first step in Phase 3 is to develop the vector **z'**. Each element of **z'** is the difference between the value of the

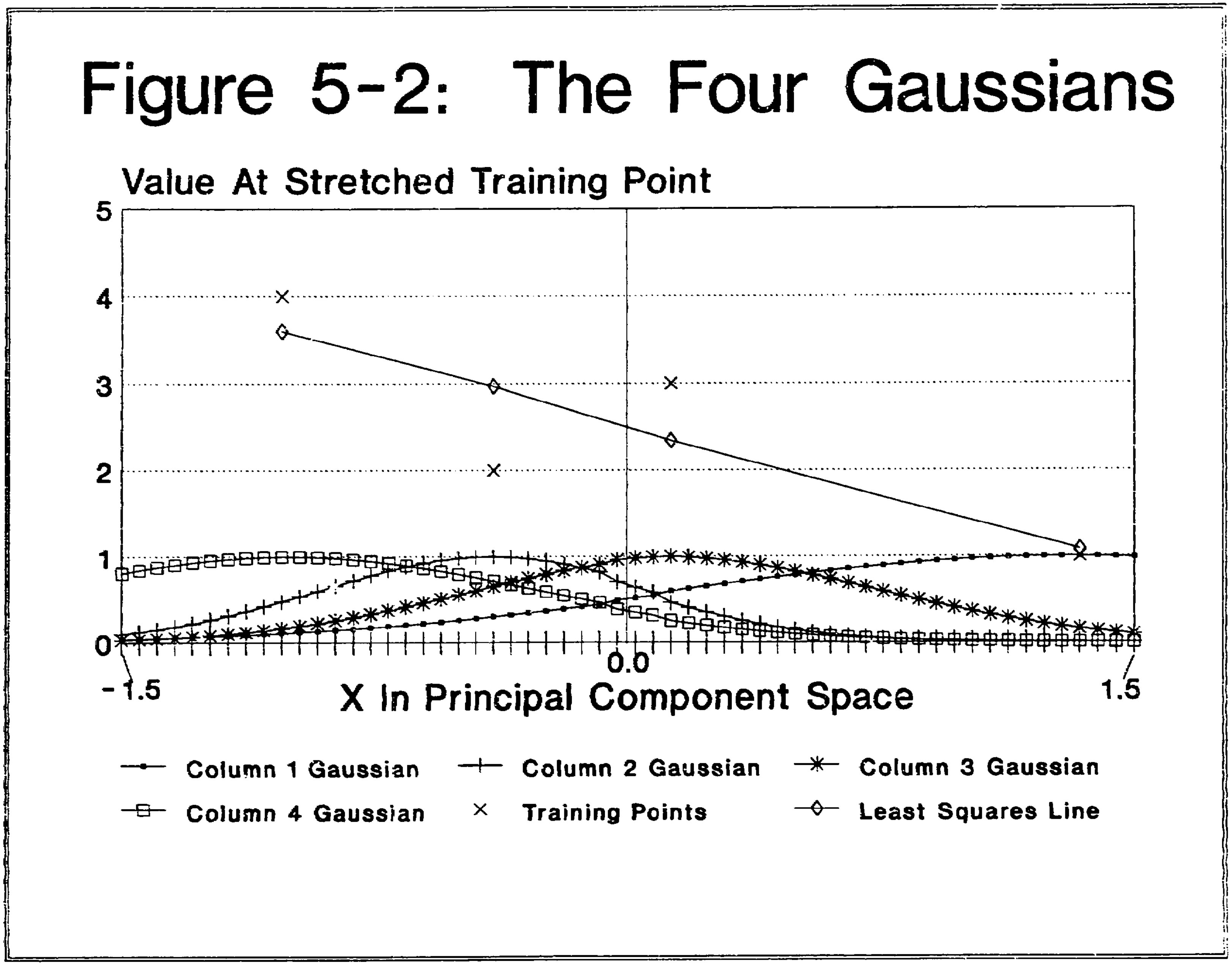

Figure 5-2. The Four Gaussians

least squares plane calculated at a training point and the value at that training point. After **a** has been resolved,

$$z' = z - Ca = z - l$$

Coefficients for the **F** matrix elements are calculated so that the sum of the previously developed Gaussian functions calculated at each training point equals the difference between the least squares plane and that training point. This task is accomplished by solving a system of m equations in m unknowns:

$$z' = Fb$$

where **b** is the vector of Gaussian coefficients, $(b_1, b_2, ..., b_m)^T$.

Since **F** is guaranteed to be nonsingular, singular value decomposition need not be used to resolve **b**. We use LU Decomposition as implemented by Press, et al. Figure 5-3 shows the modified Gaussians from our continuing example, where **z'** and **b** were determined as follows:

$$\mathbf{z'} = \begin{array}{r} -0.086 \\ -0.971 \\ 0.657 \\ 0.400 \end{array} \qquad \mathbf{b} = \begin{array}{r} -0.484 \\ -3.632 \\ 2.242 \\ 1.873 \end{array}$$

Remember that each **F** matrix column refers to the Gaussian surrounding a specific training point. The **z'** modification of those Gaussians permits their summation at the training points to equal the difference between the least squares plane and those points. Figure 5-4 shows the summation of the **z'** modified Gaussians.

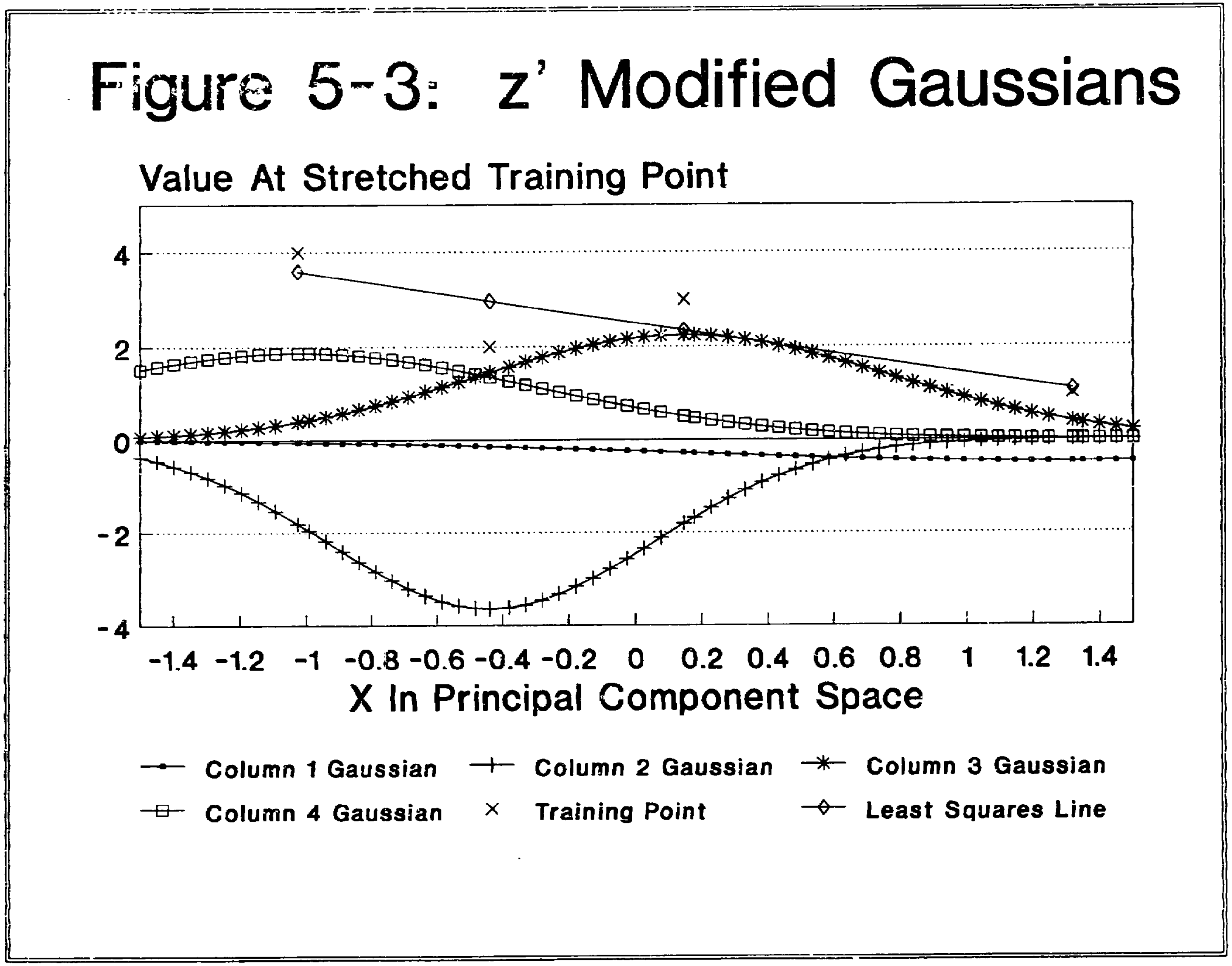

Figure 5-3. z' Modified Gaussians

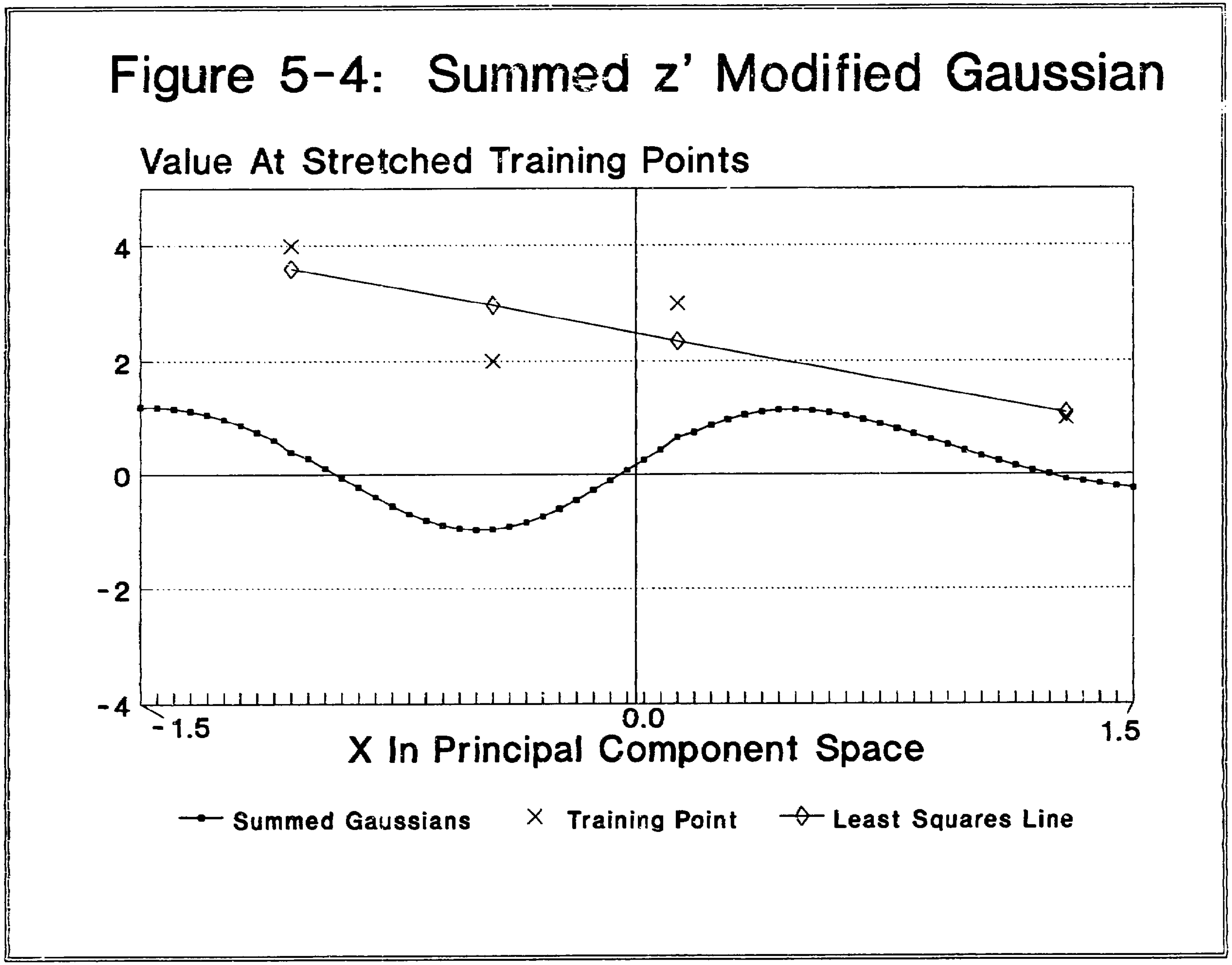

Figure 5-4. Summed z' Modified Gaussians

The last step in Phase 3 is to complete the "hammering" by combining the sum of the **z'** modified Gaussians with the least squares plane. The resulting vector, **z"**, contains values on a continuous surface at the training points' coordinates. This surface can be used for interpolation after **a** and **b** have been resolved. Figure 5-5 shows **z"** calculated for our continuing example. Note that **z"** does indeed contact each training point.

$$z'' = Fb + Ca = z' + l$$

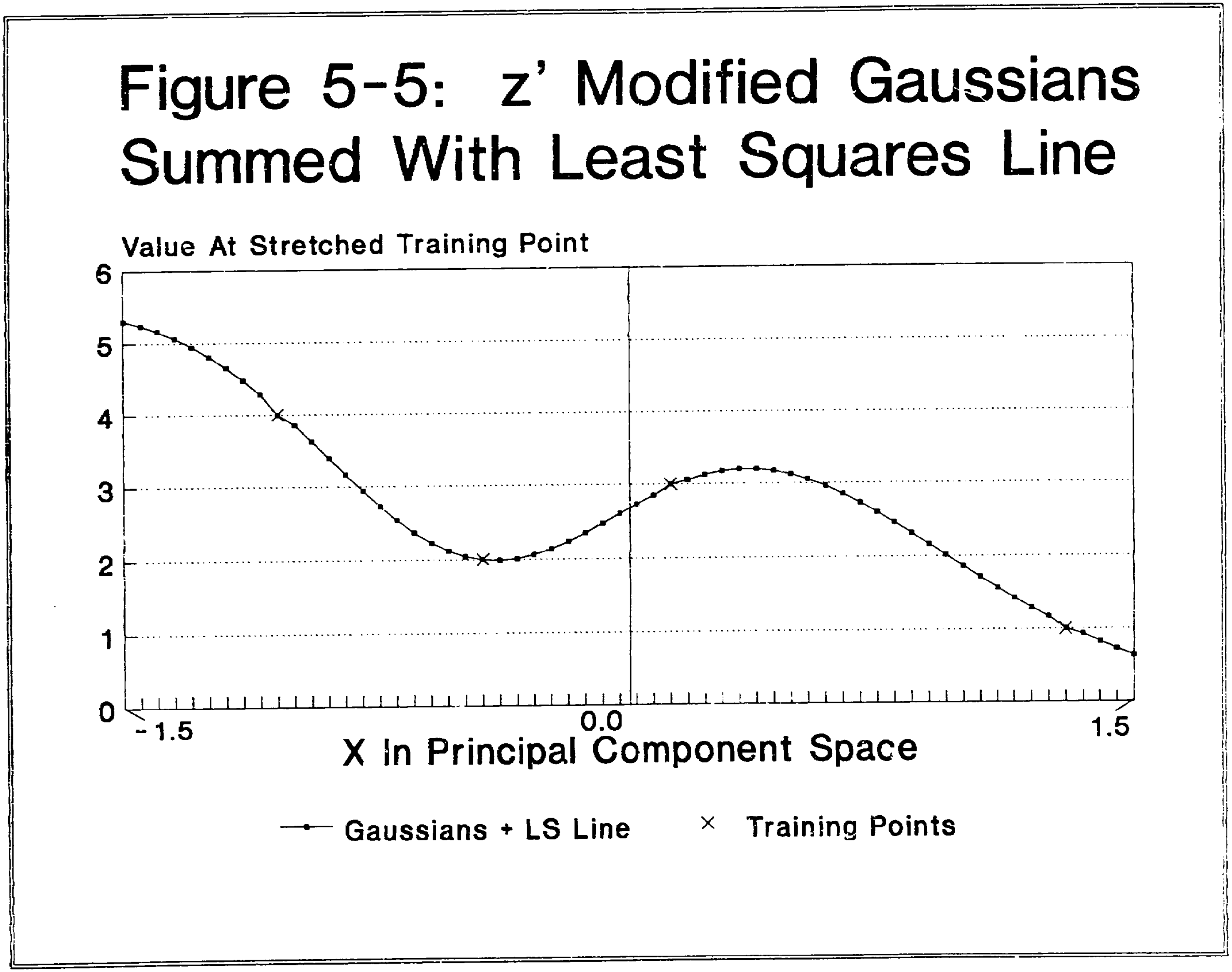

Figure 5-5. z' Modified Gaussians Summed with Least Squares Line

6. Execution For Testing

The function that the "hammering" process develops to exactly fit the training points is continuous. Thus, any point that has the same dimensionality as the hyper-space generated by the network may be chosen and an output calculated.

Typically, test and training points are chosen in the original problem space. If the network was trained in principal component space, the coordinates of the test points must be mapped from the original space to principal component space. The following equation performs this mapping:

$$p = B(o - \bar{x})$$

where: p is the test vector in principal component space
o is the test vector in original space
B is the transformation matrix calculated as part of the principal components analysis during the "stretching" portion of training
x-bar is the vector of element means
(In p and o, each element contains a coordinate value for a given dimension)

Note that if matrix **B** is the identity matrix **I** and x-bar is the **0** vector, then vector **p** is the same as vector **o**. These values of **B** and x-bar are used if the network has been trained in the original problem space. Given this more general understanding of **B** and x-bar, it is clear that vector **p** is simply the test vector in original space mapped to the appropriate space.

Once the test vector is in the appropriate space, the following equation generates the position on the hyper-surface developed during the "hammering" portion of SHNN training:

$$h = gb + pa + a_0$$

where:
- b is the vector of Gaussian coefficients
- a is the vector of least squares coefficients, $(a_1, a_2, ..., a_n)^T$
- a_0 is the least squares plane z-axis intercept
- g is the row vector of Gaussian training functions calculated at the test point
- h is the scalar value of the "hammered" surface at the coordinates of the test point
- p is the test vector mapped to the appropriate space and taken as a row vector

Figure 6-1 shows this equation calculated for our continuing example. Here, 60 test points were chosen in the range -5.0 through +5.0. h was then plotted at the coordinates indicated by the test vector elements. Note that the SHNN extrapolates asymptotically to the least squares plane as the distance from the edge of the training domain increases.

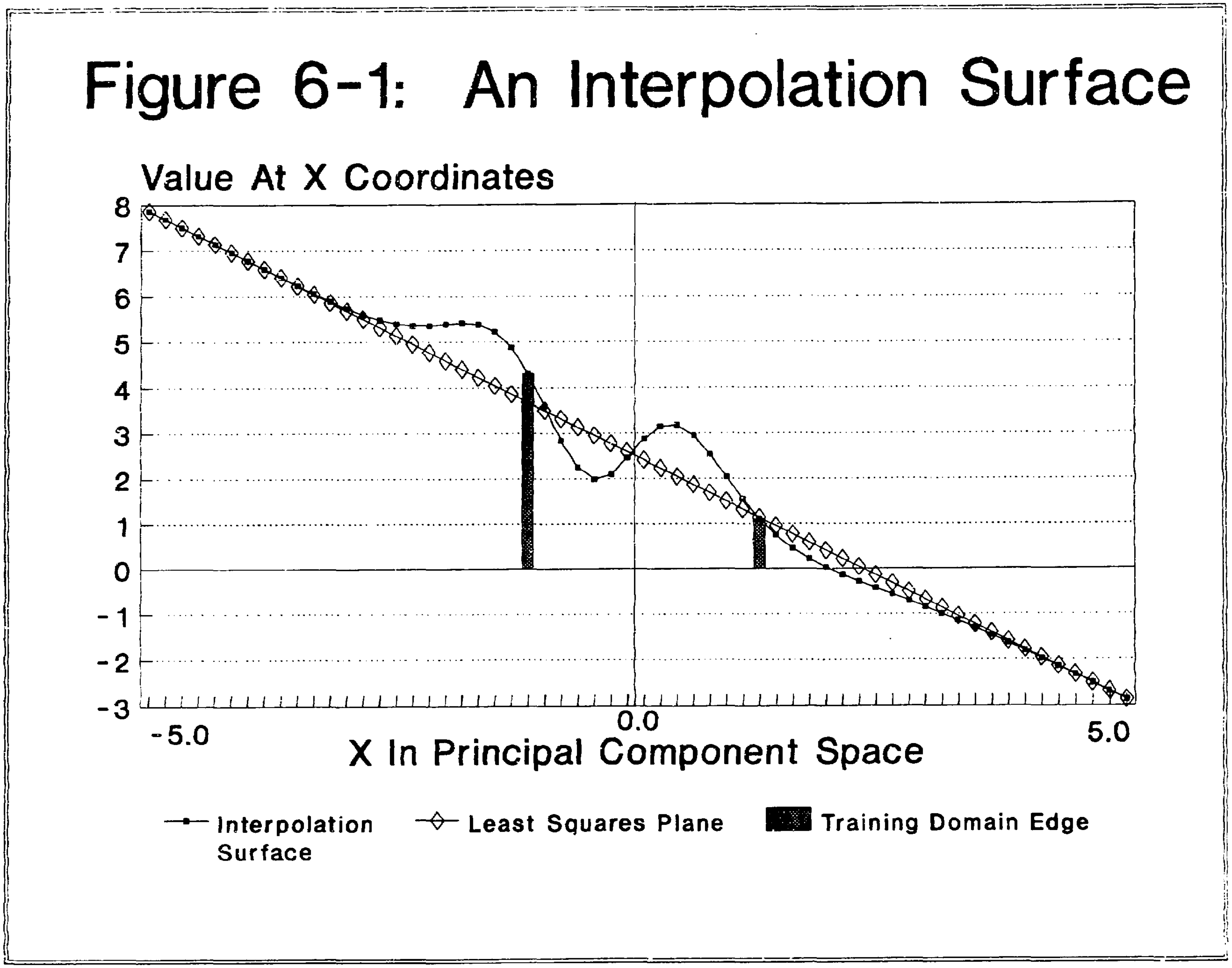

Figure 6-1. An Interpolation Surface

7. Interpretation as a Neural Network

The **B** matrix and the **a** and **b** vectors developed during SHNN training can be viewed as neural network weights. The test vector in the original space would be the network input. This section discusses the resulting SHNN neural network architecture.

As indicated in Figure 7-1, the SHNN architecture connects every node in a lower level to every node in the layer above. The nodes in a given layer are not connected to other nodes in that layer. Data flows through the network with the input at the bottom of the figure and the output at the top. Weight values are placed along the inter-node connection lines and are subscripted to show that the data flowing on the line to the input of one node from the output of another are modified. The two-letter subscript's first letter is the "to" node and the second letter is the "from" node. Each node typically executes a weighted summation of its inputs and feeds the result directly to its output. In this case, no transfer or output functions are employed making for a very simple node. In nodes that occur less often in the architecture, a transfer function also is used.

The equations discussed in Section 6, "Execution for Testing," lend themselves in a natural way to parallel implementation. Figure 7-2 shows an overview. There are two values which are summed to obtain the SHNN output, the value on the Gaussian curve, **gb**, and the value on the least squares plane, $\mathbf{a}_0$ + **pa**, of the test vector coordinates, **o**, in original space. A weight of 1 on each line from the input layer to the single-node output layer

Figure 7-1: SHNN Architecture Description of Network Notation

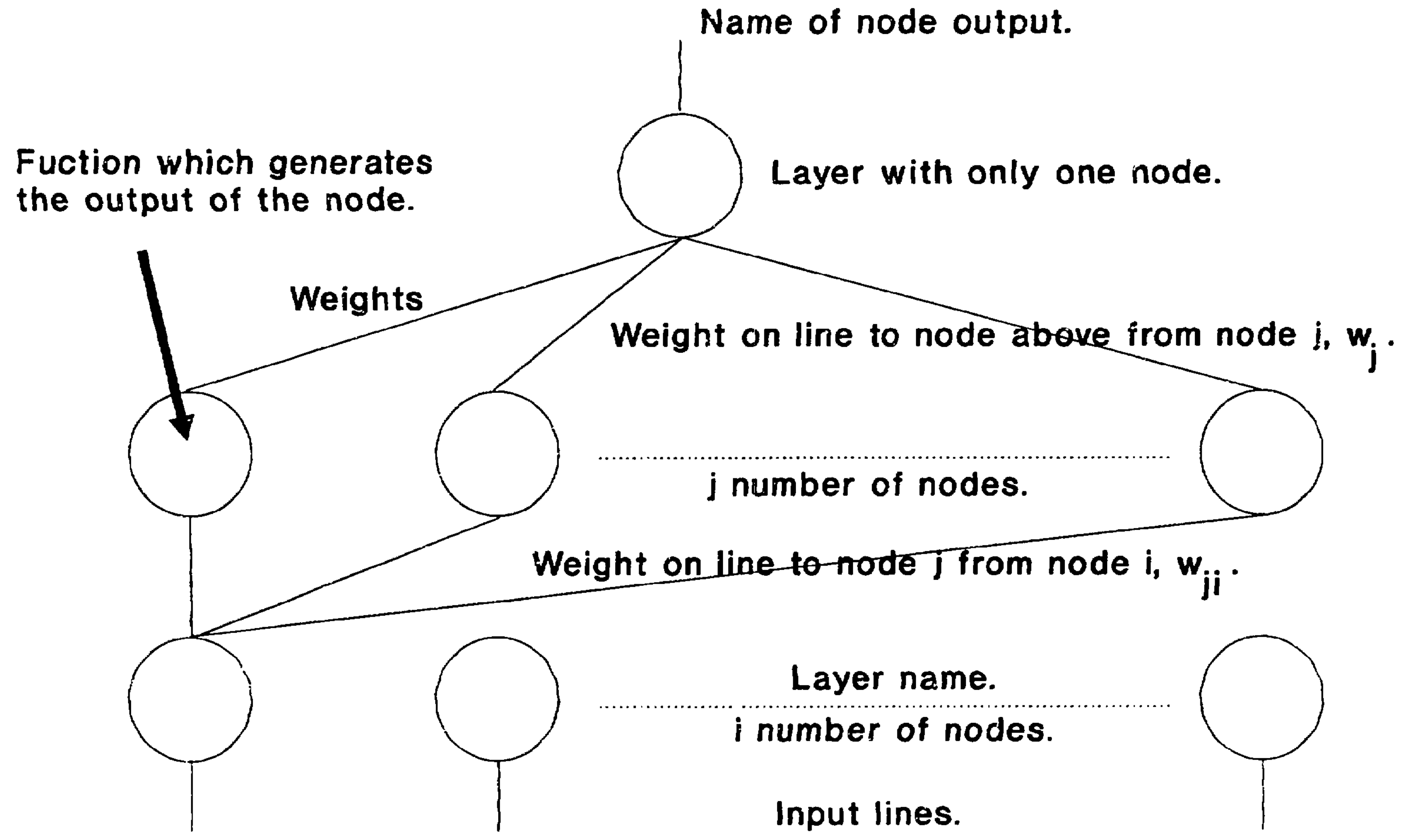
Name of node output.
Fuction which generates the output of the node.
Layer with only one node.
Weights
Weight on line to node above from node j, w_j.
j number of nodes.
Weight on line to node j from node i, w_{ji}.
Layer name.
i number of nodes.
Input lines.

Figure 7-1. SHNN Architecture. Description of Network Notation

Figure 7-2: SHNN Architecture - Overview

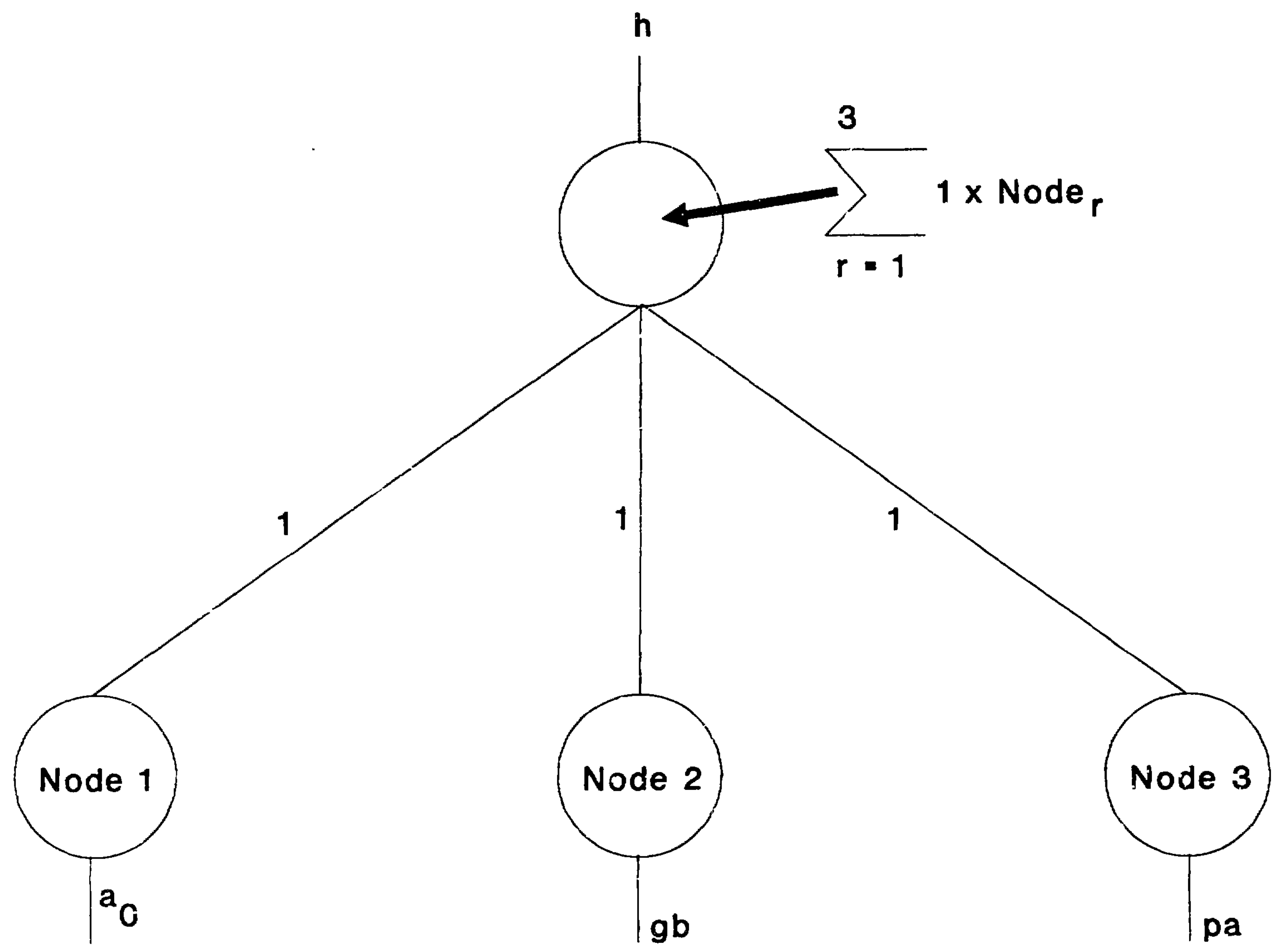

Figure 7-2. SHNN Architecture. Overview

indicates that the input layer serves only to distribute the input lines to the layer above and that the layer performs an input function that is only an unweighted summation.

The z-axis intercept, $\mathbf{a}_0$, is calculated during training and needs no further development. Figure 7-3 contains the architecture for determining the rest of the value on the least squares plane. This architecture also maps the test vector from original space to principal component space. The test vector is input at the bottom of the figure. The hidden layer produces **p**, the test vector in principal component space. The output layer result is **pa**. The weights on the lines to the hidden layer from the input layer are the element values of the transformation matrix **B**. Remember, **B** is an n-x-n matrix, where n is the number of elements in the test and training vectors. $\mathbf{B}_{ji}$ is row j, element i, of matrix **B**. This corresponds to the weight on the line to hidden node j from input node i. The weights $\mathbf{a}_j$ on the lines going to the output layer from the hidden layer are the least squares coefficients assigned to the corresponding elements of **p**.

Figure 7-4 gives the architecture for determining the position of **p** on the Gaussian surface, **gb**. Vector **p** is input at the bottom of the figure. The input layer, in this case, simply serves to distribute the input to the hidden layer nodes. Thus, the lines to the hidden layer from the input layer are weighted at 1. The hidden layer has one node for each of the training examples. The memory for each hidden layer node holds a unique training vector, $\mathbf{u}_k$, and its associated variance multiplied by -2 ($-2\sigma_k^2$, -2 x Sigma^2_k in the figures). Each hidden layer node executes an input unweighted

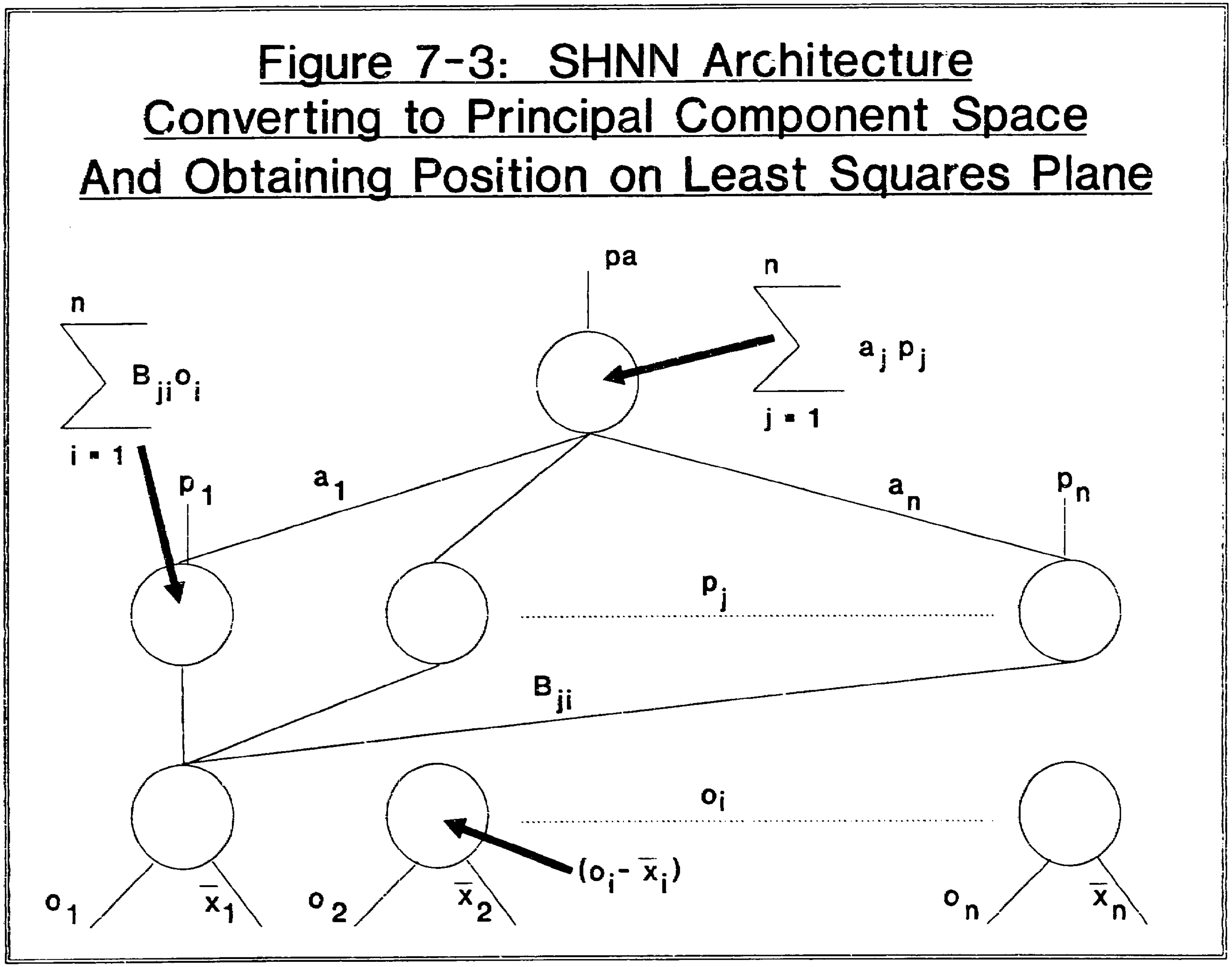

Figure 7-3. SHNN Architecture. Converting to Principal Component Space and Obtaining Position on Least Squares Plan

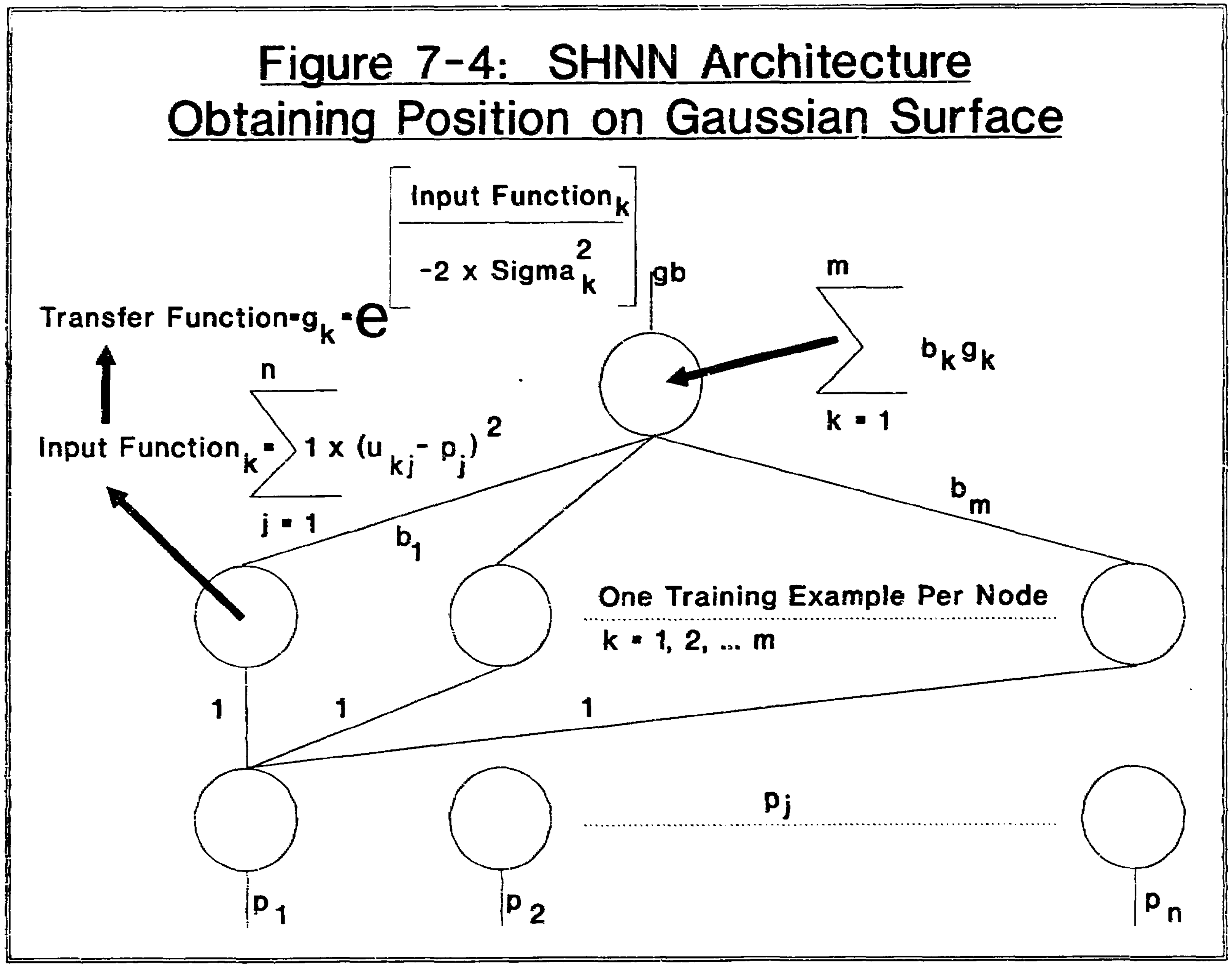
Figure 7-4: SHNN Architecture
Obtaining Position on Gaussian Surface
Transfer Function $= g_k = e^{\left[\frac{\text{Input Function}_k}{-2 \times \text{Sigma}_k^2}\right]}$
Input Function$_k = \sum_{j=1}^{n} 1 \times (u_{kj} - p_j)^2$
gb
$\sum_{k=1}^{m} b_k g_k$
b_1
b_m
One Training Example Per Node
k = 1, 2, ... m
1
1
1
p_j
p_1
p_2
p_n

Figure 7-4. SHNN Architecture. Obtaining Position on Gaussian Surface

summation whose result is used to drive a Gaussian transfer function. The hidden layer is connected to the output layer. The weights between these layers are the Gaussian coefficients, **b**, related to a specific training vector. The output layer executes a weighted summation of the hidden layer output to obtain **gb**.

8. Application to 3D Spaces

This section presents an example of the SHNN applied to a small three-dimensional problem. Figures are presented which show input vectors moving through training and testing. Note that in the case of 3D spaces two dimensions are used for independent variables and the third dimension is used for the dependent variable. Accordingly, n (the number of elements or coordinates in the training and test vectors) is equal to 2. A subscript convention employed refers to matrix elements $\mathbf{M}_{ij}$, where the first subscript refers to the matrix row and the second subscript refers to the matrix column.

Stretching

Consider a matrix of training vectors in original space. The rows are the m vectors and the columns are the n vector elements or coordinates.

$$X = \begin{pmatrix} X_{11}, X_{12}, \ldots, X_{1n} \\ X_{21}, X_{22}, \ldots, X_{2n} \\ \cdot \\ \cdot \\ \cdot \\ X_{m1}, X_{m2}, \ldots, X_{mn} \end{pmatrix}$$

For our example, this matrix reduces as follows. The vector **z** contains the expected SHNN output for each training vector. A plot illustrating this set of training vectors is given in Figure 8-1.

$$X = \begin{pmatrix} 2.3, & 1.1 \\ 3.0, & 3.0 \\ 3.0, & 4.0 \\ 5.5, & 6.2 \end{pmatrix} \qquad z = \begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{pmatrix}$$

The vector of column means is:

$$\bar{x} = \begin{pmatrix} (X_{11} + X_{21} + \dots + X_{m1})/m \\ (X_{12} + X_{22} + \dots + X_{m2})/m \\ \cdot \\ \cdot \\ \cdot \\ (X_{1n} + X_{2n} + \dots + X_{mn})/m \end{pmatrix}$$

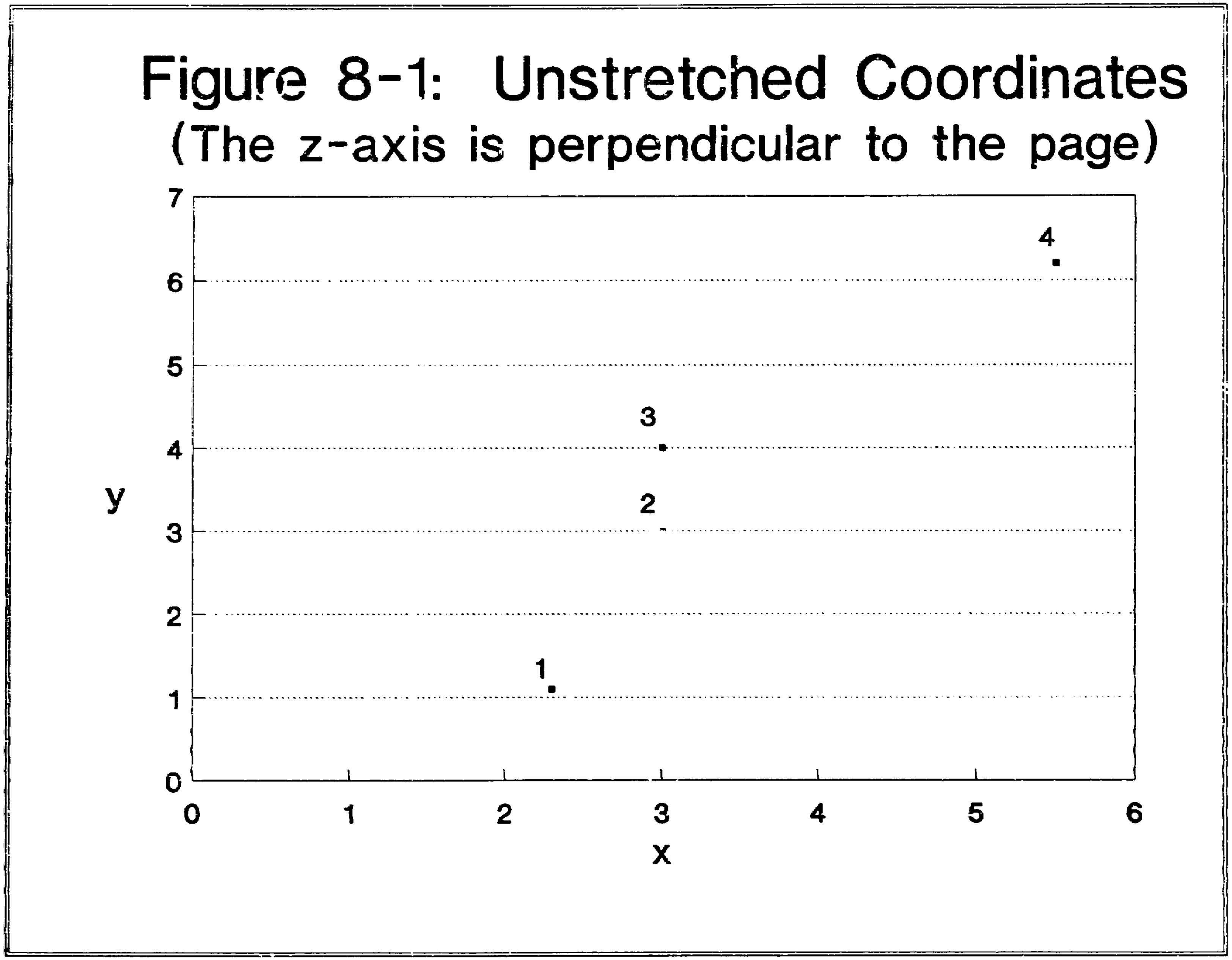

Figure 8-1. Unstretched Coordinates

For this example, the following means were calculated.

$$\bar{x} = \begin{pmatrix} 3.450 \\ 3.575 \end{pmatrix}$$

The matrix of input elements relative to their means is:

$$X' = \begin{pmatrix} X_{11} - \bar{x}_1, X_{12} - \bar{x}_2, \ldots, X_{1n} - \bar{x}_n \\ X_{21} - \bar{x}_1, X_{22} - \bar{x}_2, \ldots, X_{2n} - \bar{x}_n \\ \cdot \\ \cdot \\ \cdot \\ X_{m1} - \bar{x}_1, X_{m2} - \bar{x}_2, \ldots, X_{mn} - \bar{x}_n \end{pmatrix}$$

This equation produces the following matrix for our example.

$$X' = \begin{pmatrix} -1.150, & -2.475 \\ -0.450, & -0.575 \\ -0.450, & 0.425 \\ 2.050, & 2.625 \end{pmatrix}$$

The corresponding covariance matrix is: (See Lipschutz for a basic discussion on matrix operations.)

$$A = \begin{pmatrix} X'_{11}, X'_{21}, \ldots, X'_{m1} \\ X'_{12}, X'_{22}, \ldots, X'_{m2} \\ \cdot \\ \cdot \\ \cdot \\ X'_{1n}, X'_{2n}, \ldots, X'_{mn} \end{pmatrix} \begin{pmatrix} X'_{11}, X'_{12}, \ldots, X'_{1n} \\ X'_{21}, X'_{22}, \ldots, X'_{2n} \\ \cdot \\ \cdot \\ \cdot \\ X'_{m1}, X'_{m2}, \ldots, X'_{mn} \end{pmatrix} \frac{1}{m-1}$$

From this, it is plain to see that the elements of matrix **A** are computable without actually creating the transpose of **X**. Rather, the following equation may be used:

$$A_{ij} = \frac{\sum_{k=1}^{n} X'_{ik} X'_{jk}}{m-1}$$

For this example, matrix **A** becomes:

$$A = \begin{pmatrix} 1.977, 2.765 \\ 2.765, 4.509 \end{pmatrix}$$

Find the matrix of eigenvectors, V, and the vector of eigenvalues, **λ**. A discussion of the required process is beyond the scope of this report. Press et al. give an excellent discussion on practical means of finding the following eigenvectors and eigenvalues. (Note: The eigenvectors are the V matrix columns.)

$$V = \begin{pmatrix} 0.842, 0.540 \\ -0.540, 0.842 \end{pmatrix}$$

$$\lambda = (0.202, 6.284)$$

Find the $\mathbf{u}_i$'s, the m training vectors mapped from original space to principal component space.

$$u_i = \begin{pmatrix} B_{11}, B_{12}, \ldots, B_{1n} \\ B_{21}, B_{22}, \ldots, B_{2n} \\ \cdot \\ \cdot \\ \cdot \\ B_{n1}, B_{n2}, \ldots, B_{nn} \end{pmatrix} \begin{pmatrix} X'_{i1} \\ X'_{i2} \\ \cdot \\ \cdot \\ \cdot \\ X'_{in} \end{pmatrix}$$

The matrix elements $\mathbf{B}_{qj}$ are calculated as individual elements of an n-x-n matrix.

$$B = \begin{pmatrix} \dfrac{V_{11}}{|V_1|\sqrt{\lambda_1}}, & \dfrac{V_{21}}{|V_1|\sqrt{\lambda_1}}, & \ldots, & \dfrac{V_{nl}}{|V_1|\sqrt{\lambda_1}} \\ \dfrac{V_{12}}{|V_2|\sqrt{\lambda_2}}, & \dfrac{V_{22}}{|V_2|\sqrt{\lambda_2}}, & \ldots, & \dfrac{V_{n2}}{|V_2|\sqrt{\lambda_2}} \\ & \cdot & & \\ & \cdot & & \\ & \cdot & & \\ \dfrac{V_{1n}}{|V_n|\sqrt{\lambda_1}}, & \dfrac{V_{2n}}{|V_n|\sqrt{\lambda_n}}, & \ldots, & \dfrac{V_{nn}}{|V_n|\sqrt{\lambda_n}} \end{pmatrix}$$

where:

$$|V_i| = \sqrt{\sum_{j=1}^{n} V_{ji}^2}$$

For our example, the transformation matrix **B** becomes:

$$B = \begin{pmatrix} 1.874, & -1.203 \\ 0.216, & 0.336 \end{pmatrix}$$

As a result of these "stretching" calculations, the original set of training vectors is mapped as shown below and plotted in Figure 8-2.

$$U = \begin{pmatrix} 0.822, & -1.079 \\ -0.152, & -0.290 \\ -1.354, & 0.046 \\ 0.684, & 1.323 \end{pmatrix} \qquad z = \begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{pmatrix}$$

where the rows of matrix $\mathbf{U} = \mathbf{u}_i^T$.

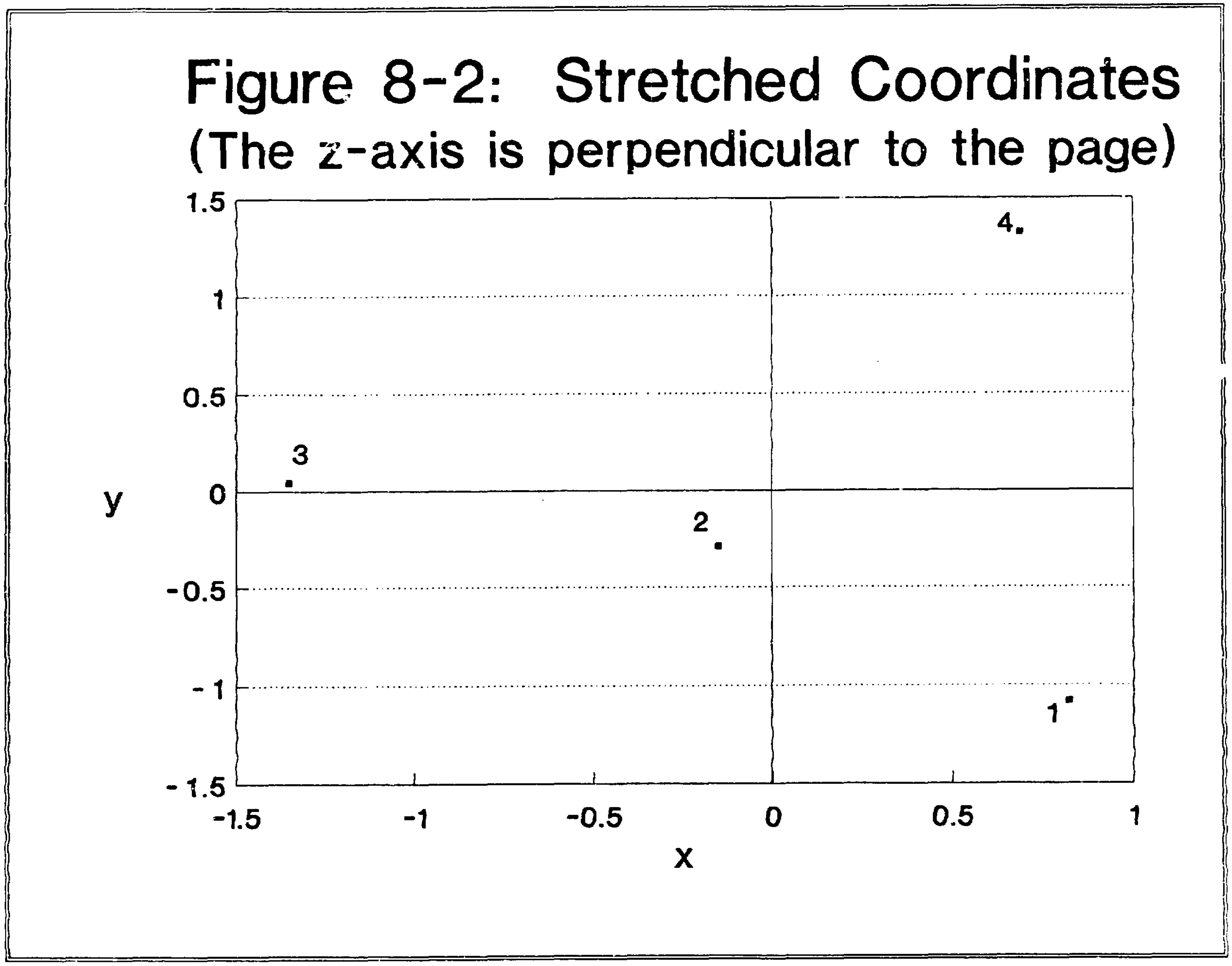

Figure 8-2. Stretched Coordinates

Hammering

Take a matrix of training vectors in original or principal component space along with the corresponding vector of expected SHNN outputs.

$$U = \begin{pmatrix} U_{11}, U_{12}, \ldots, U_{1n} \\ U_{21}, U_{22}, \ldots, U_{2n} \\ \cdot \\ \cdot \\ \cdot \\ U_{m1}, U_{m2}, \ldots, U_{mn} \end{pmatrix} \qquad z = \begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_m \end{pmatrix}$$

For our example, we will continue with the previous training vectors in principal component space and the same expected SHNN outputs.

Fit a least squares plane to the training data. This involves solving an overspecified, ($m > n$), system of equations.

$$\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_m \end{pmatrix} = \begin{pmatrix} 1, U_{11}, U_{12}, \ldots, U_{1n} \\ 1, U_{21}, U_{22}, \ldots, U_{2n} \\ \cdot \\ \cdot \\ \cdot \\ 1, U_{m1}, U_{m2}, \ldots, U_{mn} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{pmatrix}$$

Our example has this system:

$$\begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{pmatrix} = \begin{pmatrix} 1.0, & 0.822, & -1.079 \\ 1.0, & -0.152, & -0.290 \\ 1.0, & -1.354, & 0.046 \\ 1.0, & 0.684, & 1.323 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}$$

The solution to this system is plotted in Figure 8-3 and has the following **a** vector:

$$a = \begin{pmatrix} 2.500 \\ -0.270 \\ 1.257 \end{pmatrix}$$

Center Gaussian functions, G($\mathbf{u}_i$,$\mathbf{u}_j$), at each of the training points. These functions are placed in matrix **F**. The columns represent training points and the rows represent G($\mathbf{u}_i$,$\mathbf{u}_j$)

Figure 8-3: Least Squares Plane with Training Points

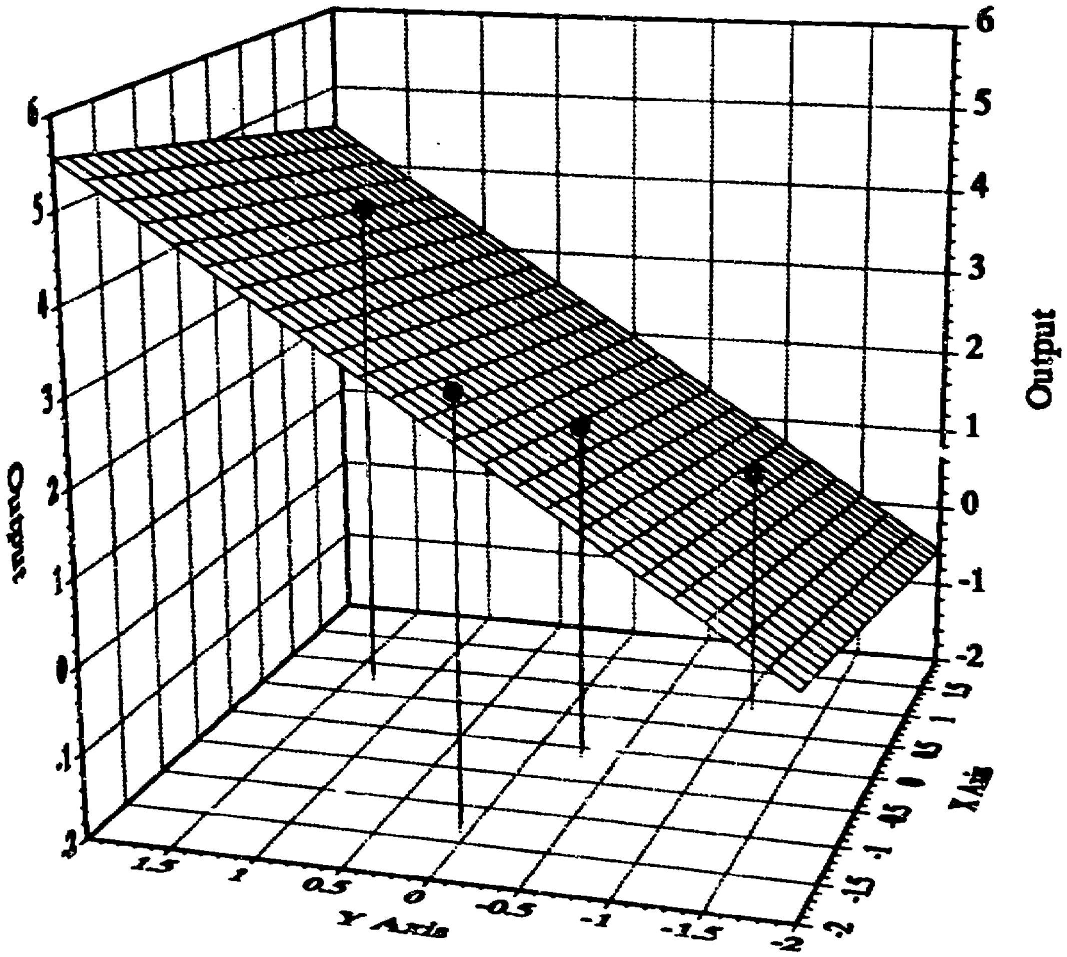

Figure 8-3. Least Squares Plane with Training Points

at the column's training point, $\mathbf{u}_i$, calculated at the coordinates of the row training point. $\mathbf{u}_j$.

$$F = \begin{pmatrix} G(u_1,u_1),\ G(u_2,u_1),\ \ldots,\ G(u_m,u_1) \\ G(u_1,u_2),\ G(u_2,u_2),\ \ldots,\ G(u_m,u_2) \\ \cdot \\ \cdot \\ \cdot \\ G(u_1,u_m),\ G(u_2,u_m),\ \ldots,\ G(u_m,u_m) \end{pmatrix}$$

where:	$\mathbf{u}_i$ refers to the training point i
	$G(\mathbf{u}_j,\mathbf{u}_k)$ is the Gaussian function for training vector j calculated at training vector k

$$G(u_j,u_k) = e^{\frac{-\sum_{i=1}^{n}(u_{ji}-u_{ki})^2}{2\sigma_j^2}}$$

where:	u_{ij}	= training element i of training vector j
	σ^2_j	= the Gaussian variance associated with the jth training point
	e	= the base of natural logarithms

Adjust the **σ**'s until **F** achieves columnar diagonal dominance. For our example, the following **F** matrix was produced. The summed Gaussian curve is compared to the least squares plane in Figure 8-4.

$$F = \begin{pmatrix} 1.000, & 0.418, & 0.176, & 0.265 \\ 0.635, & 1.000, & 0.637, & 0.469 \\ 0.177, & 0.421, & 1.000, & 0.265 \\ 0.188, & 0.160, & 0.187, & 1.000 \end{pmatrix}$$

$$2\sigma^2 = (3.459,\ 1.801,\ 3.451,\ 4.359)$$

Force the peaks of the Gaussian functions to equal the difference between the least squares plane and the training points. This is accomplished by solving another system of equations. First, develop **z'**, the vector of differences between the training points and the least squares plane.

$$z' = \begin{pmatrix} z_1 - a_0 - a_1u_{11} - a_2u_{12} - \ldots - a_nu_{1n} \\ z_2 - a_0 - a_1u_{21} - a_2u_{22} - \ldots - a_nu_{2n} \\ \cdot \\ \cdot \\ \cdot \\ z_m - a_0 - a_1u_{m1} - a_2u_{m2} - \ldots - a_nu_{mn} \end{pmatrix}$$

Figure 8-4: Least Squares Plane, Sum of Gaussian Basis Functions, and Training Points

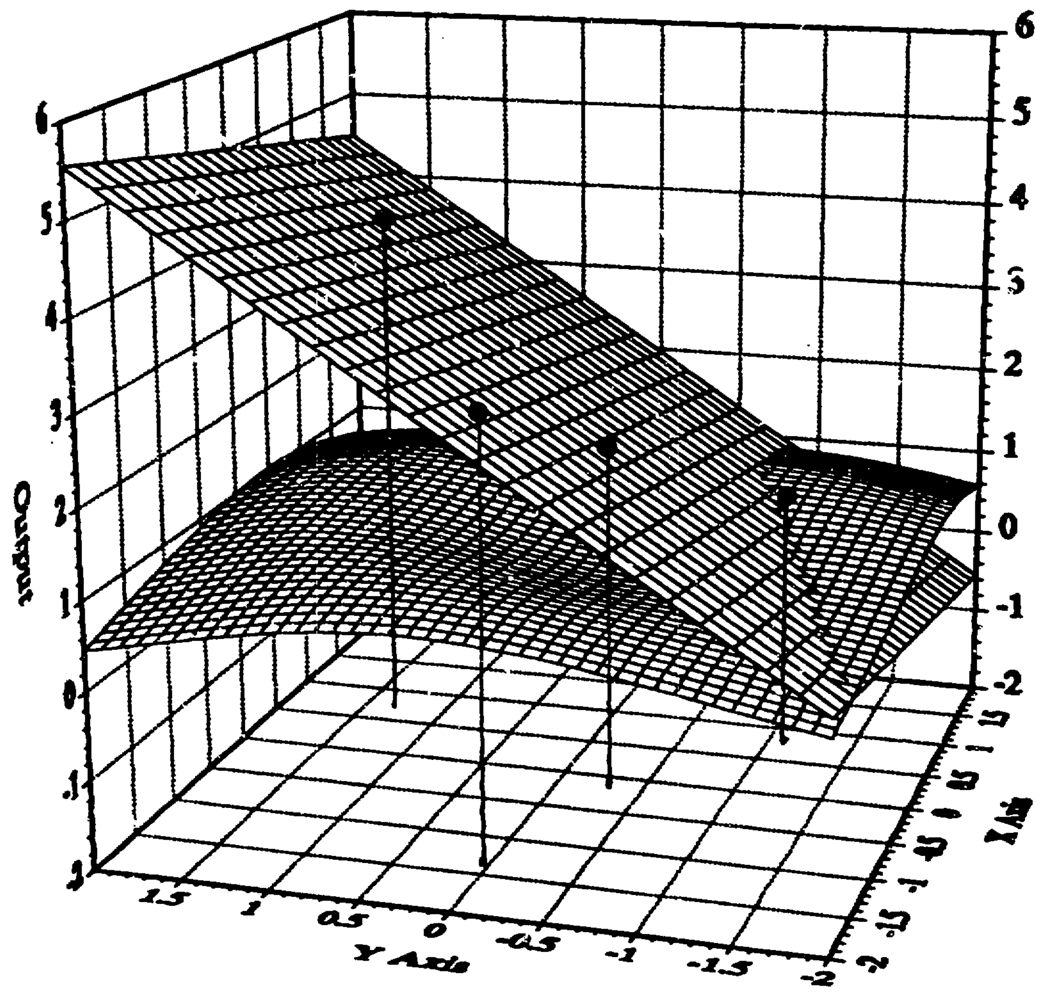

Figure 8-4. Least Squares Plane, Sum of Gaussian Basis Functions, and Training Points

The example produced the following **z'** vector:

$$z' = \begin{pmatrix} 0.077 \\ -0.176 \\ 0.078 \\ 0.022 \end{pmatrix}$$

Use **z'** to solve for a set of coefficients which will adjust the Gaussian peaks so that they represent the difference between the expected SHNN output at the training points and the value of the least squares plane at those points.

$$\begin{pmatrix} z'_1 \\ z'_2 \\ \cdot \\ \cdot \\ \cdot \\ z'_m \end{pmatrix} = \begin{pmatrix} F_{11}, F_{12}, \ldots, F_{1m} \\ F_{21}, F_{22}, \ldots, F_{2m} \\ \cdot \\ \cdot \\ \cdot \\ F_{m1}, F_{m2}, \ldots, F_{mm} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{pmatrix}$$

Our example resolved the **b** vector as

$$b = \begin{pmatrix} 0.234 \\ -0.480 \\ 0.235 \\ 0.015 \end{pmatrix}$$

The sum of the adjusted Gaussian curves are compared to the least squares plane in Figure 8-5.

Figure 8-5: Least Squares Plane,
Sum of Gaussian Basis Functions with Coefficients,
and Training Points

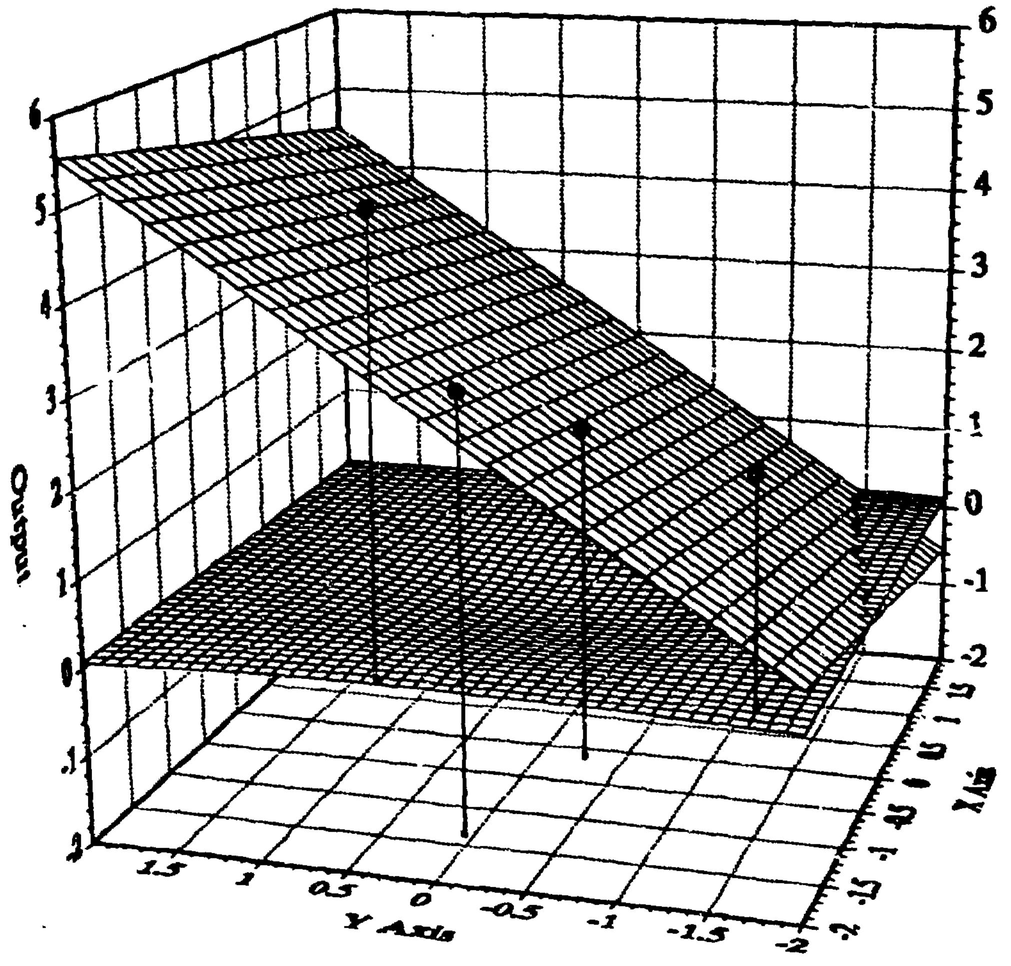

Figure 8-5. Least Squares Plane, Sum of Gaussian Basis Functions with Coefficients, and Training Points

Execution

Given the previous calculations, it is now possible to use the SHNN to calculate outputs based on test vectors not used in training. If the network was trained with the training vectors in principal component space, the test vectors must also be mapped to that space.

$$\begin{pmatrix} p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{pmatrix} = \begin{pmatrix} B_{11}, B_{12}, \ldots, B_{1n} \\ B_{21}, B_{22}, \ldots, B_{2n} \\ \cdot \\ \cdot \\ \cdot \\ B_{n1}, B_{n2}, \ldots, B_{nn} \end{pmatrix} \left[\begin{pmatrix} o_1 \\ o_2 \\ \cdot \\ \cdot \\ \cdot \\ o_n \end{pmatrix} - \begin{pmatrix} \overline{x_1} \\ \overline{x_2} \\ \cdot \\ \cdot \\ \cdot \\ \overline{x_n} \end{pmatrix} \right]$$

A single mapping for this example is shown below, assuming that the test vector $\mathbf{o} = (2.300, 1.100)^T$.

$$\begin{pmatrix} 0.822 \\ -1.079 \end{pmatrix} = \begin{pmatrix} 1.874, & -1.203 \\ 0.216, & 0.336 \end{pmatrix} \left[\begin{pmatrix} 2.300 \\ 1.100 \end{pmatrix} - \begin{pmatrix} 3.450 \\ 3.575 \end{pmatrix} \right]$$

The appropriately mapped test vectors are used to calculate the SHNN's output.

$$\textit{SHNN Output} = \left(G(p,t_1),\ G(p,t_2),\ \ldots,\ G(p,t_m)\right) \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{pmatrix} +$$

$$\left(1,\ p_1,\ \ldots,\ p_n\right) \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{pmatrix}$$

Choosing **p**, the mapped test vector as calculated above, the following SHNN result is calculated:

$$1 = (1.000,\ 0.208,\ 0.003,\ 0.003)\begin{pmatrix} 0.234 \\ -0.480 \\ 0.235 \\ 0.011 \end{pmatrix} +$$

$$(1,\ 0.822,\ -1.079)\begin{pmatrix} 2.500 \\ -0.270 \\ 1.257 \end{pmatrix}$$

Figure 8-6 shows the surface generated if a regular matrix of test vectors is chosen surrounding the test points (coordinate axes bounded by -2 through 2 inclusive).

Figure 8-6: Fully Stretched and Hammered Output and Training Points

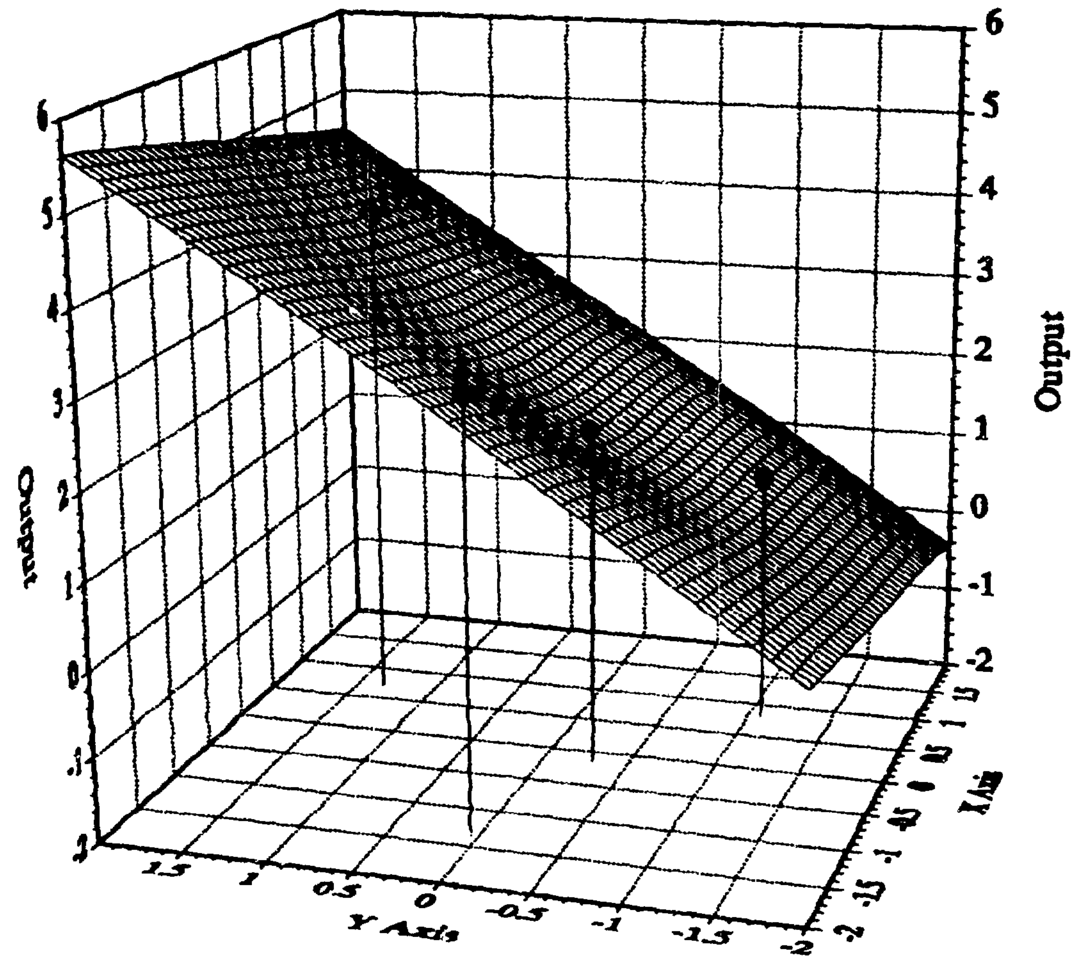

Figure 8-6. Fully Streched and Hammered Output and Training Points

9. Data for Network Training and Testing in 3D Solid Modeling

The SHNN network was trained and tested for 3D interpolation using data generated from an equation z = f(x,y) where z is height above a table or baseline and x and y are independent coordinate axis values. For example, let f(x,y) = sin(x)sin(y), x = 0.5, and y = 0.25, then z = f(x,y) = sin(x)sin(y) = sin(0.5)sin(0.25) = 0.1186118. This point is plotted in Figure 9-1.

Training data were generated from a regular sampling grid. First, the limits of x and y were defined and a grid density was specified in terms of the number of evenly spaced y coordinates required for each of a number of evenly spaced x coordinates. For example, if the limits of both x and y are each 0.0 through 1.0 and a 3-by-5 grid is selected, then there are 15 total data samples as shown in Figure 9-2. Figure 9-3 extends this example by showing sin(x)sin(y) for the original limits but using a 20-by-20 regular sampling grid. The axes are tilted to display the surface shape.

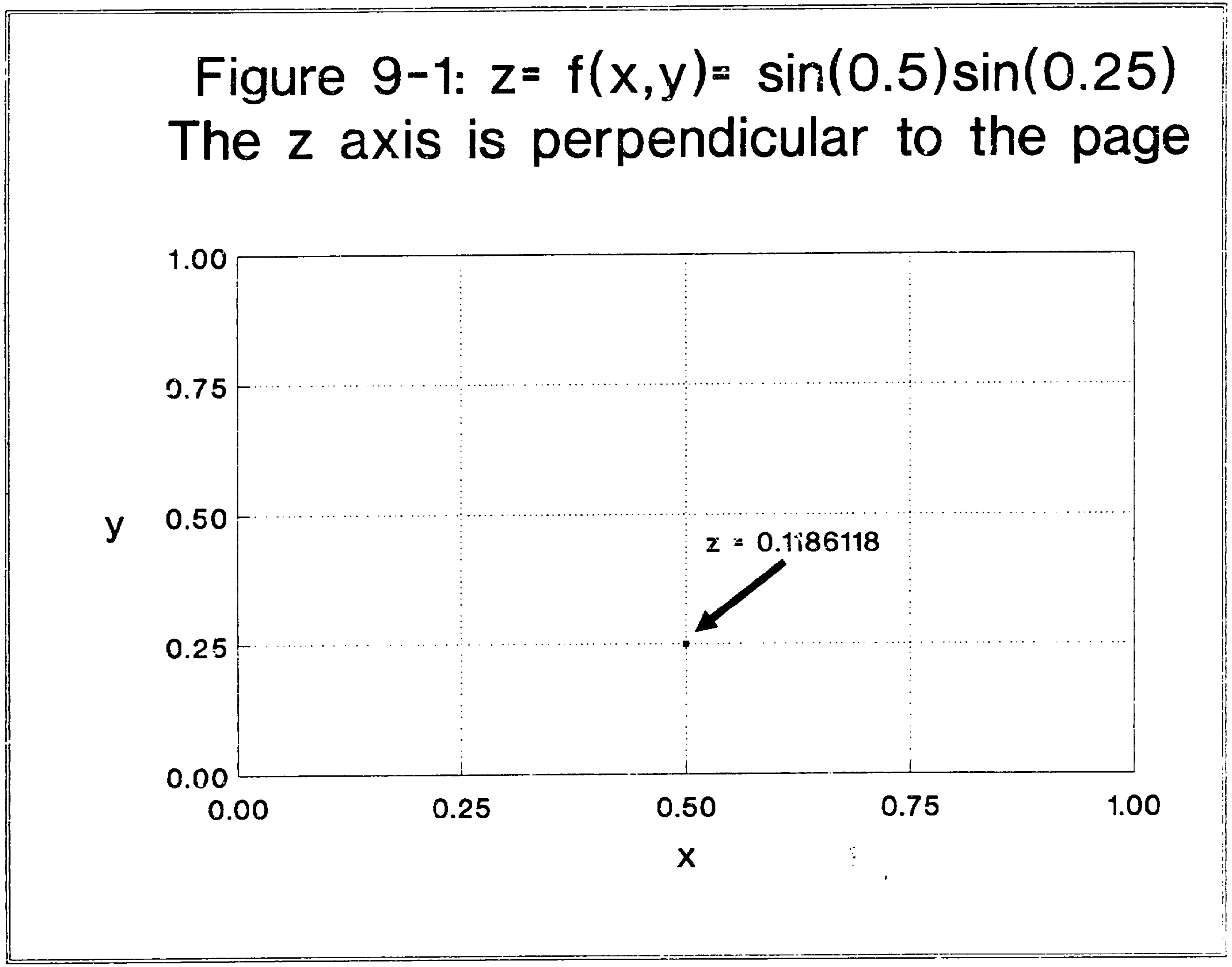

Figure 9-1. $z = f(x,y) = \sin(0.5)\sin(0.2)$

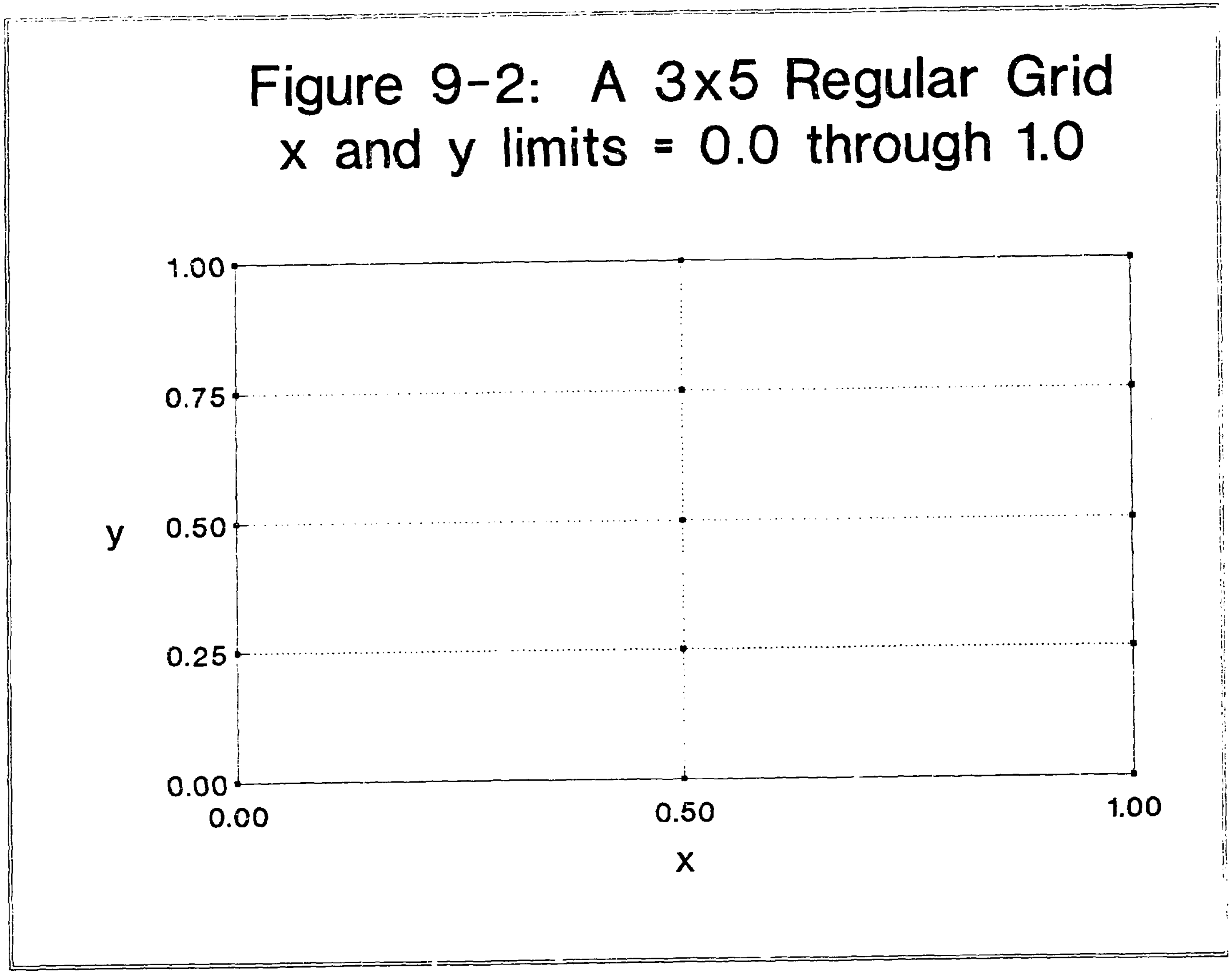

Figure 9-2. A 3x5 Regular Grid

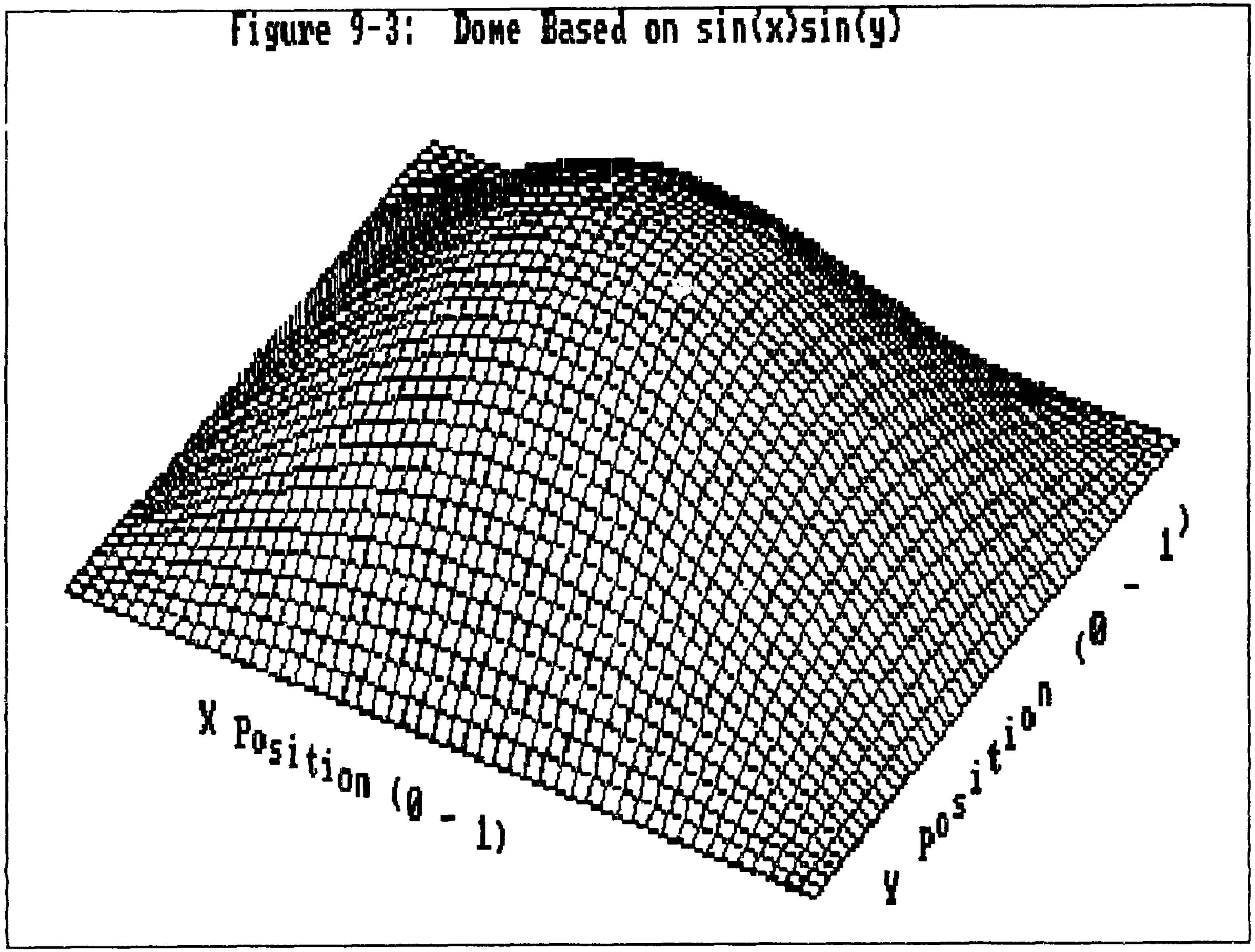

Figure 9-3. Dome Based on sin(x)sin(y)

All (x,y) training coordinates for a regular i-by-j grid were generated using the following equations:

$$x_i = \min(x) + \frac{(i - 1)(\max(x) - \min(x))}{\max(i) - 1}$$

$$y_j = \min(y) + \frac{(j - 1)(\max(y) - \min(y))}{\max(j) - 1}$$

where i = 1,2, ..., # x coordinates and j = 1,2, ..., # y coordinates for each x coordinate.

Test data were produced by positioning these data relative to the training data. The goal in test data production was to cause the maximum loss of interpolation precision. This goal was very easy to attain. It was only necessary to put the test points at the maximum distance from the training points while, like the training points, covering the entire surface. For grid sampling, the test points were placed in the center of the grid blocks formed by the training points. Figure 9-4 repeats Figure 9-2 with the test points entered. The test points lie at coordinates (x_a, y_a).

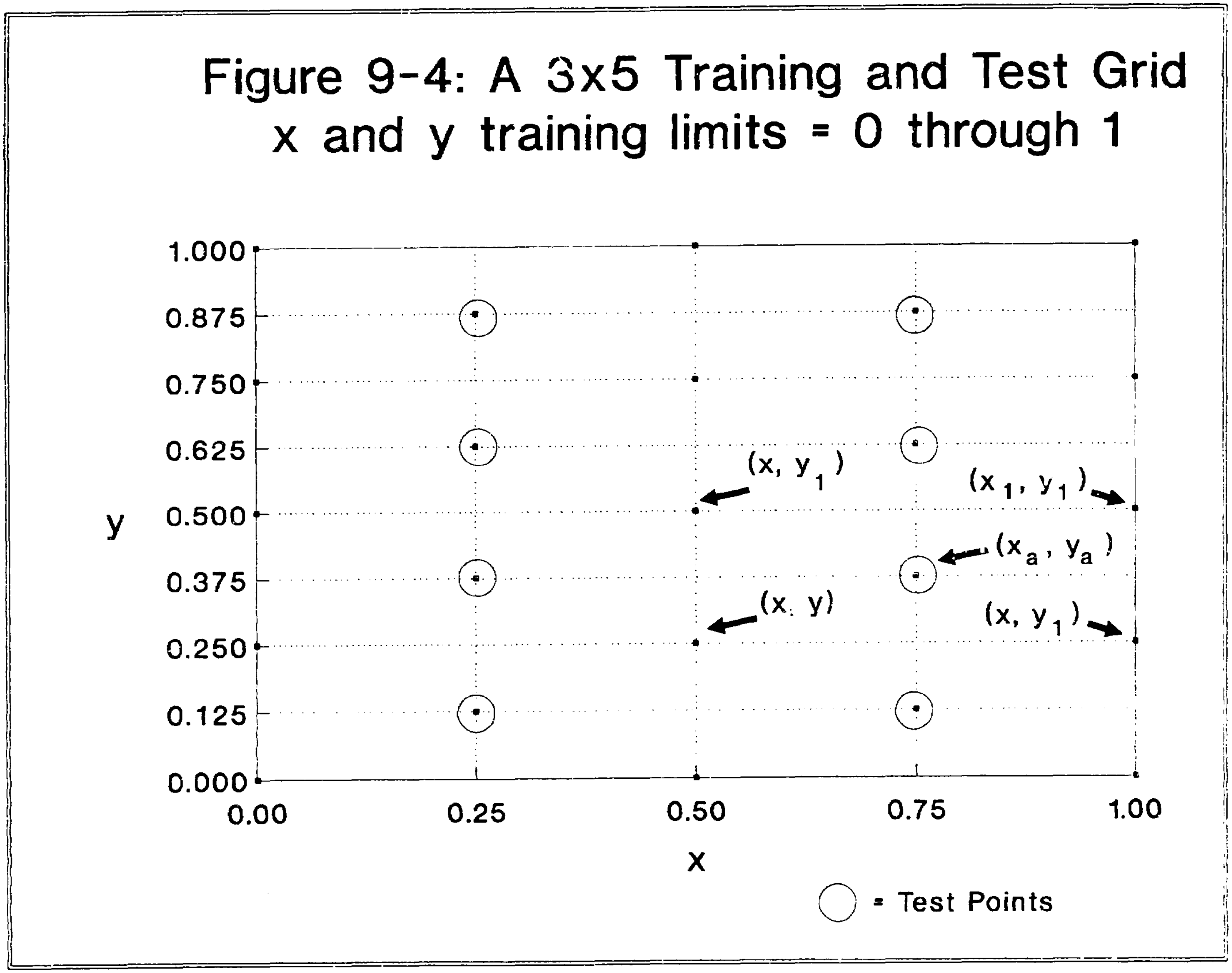

Figure 9-4. A 3x5 Training and Test Grid

In general, $x_a = x + 0.5(x_1 - x)$ and $y_a = y + 0.5(y_1 - y)$; where x, y, x_1, and y_1 are the four corners of a given grid as noted in Figure 9-4.

For regular grids $x_{a+i} = x + 0.5(x_1 - x) + i(x_1 - x)$ and $y_{a+i} = y + 0.5(y_1 - y) + j(y_1 - y)$; where $i = 0,1, ...,(t-1)$ and $j = 0,1, ...(u-1)$. If the training grid is t-by-u (that is: u y-coordinates for each of t x-coordinates) then the testing grid is (t-1)-by-(u-1), where both t and u must be equal to 2 or greater.

When the testing data were applied to the trained network, a measure of interpolation precision was calculated. This measure essentially reveals the overall difference between the answer the trained network result and the actual function result. The same training and testing data and the same measure of precision were used for both the SHNN and traditional interpolation. The equations used for precision calculations are given below.

$$Precision = \frac{CalculatedResult - InterpolatedResult}{CalculatedResultsRange}$$

$$RMS\ Precision = \sqrt{\frac{\sum Precision^2}{\#TestVectors}}$$

$$Average\ Precision = \frac{\sum |Precision|}{\#TestVectors}$$

10. Tests on Three Surfaces

The SHNN was tested on surfaces of low, medium, and high complexity. The results were compared to bicubic interpretation. Training grids of 3, 4, 5, 6, 7, 8, 9, 15, 20, 25, and 31 were chosen. The available computing hardware did not permit higher densities. The training and test points were chosen as described in "Data For Network Training and Testing in 3D Solid Modeling."

Low Complexity Surface

The surface of low complexity is described by the following equation:

$$\textit{Surface Height} = \cos\left(\frac{\pi x}{2}\right)\left(\frac{1-\left(\sqrt{e}\, e^{\frac{-y^2}{2}}\right)}{1-\sqrt{e}}\right)$$

A plot of this equation is shown in Figure 10-1 for a 50x50 grid with the x and y axes in the range of 0 through 1 inclusive. This equation was chosen because it is not symmetric on either coordinate axis. Thus, there are no duplicated test points.

Figure 10-2 compares the RMS precision of the SHNN to that of bicubic interpolation. Figure 10-3 compares the worst precision of the SHNN to that of bicubic interpolation.

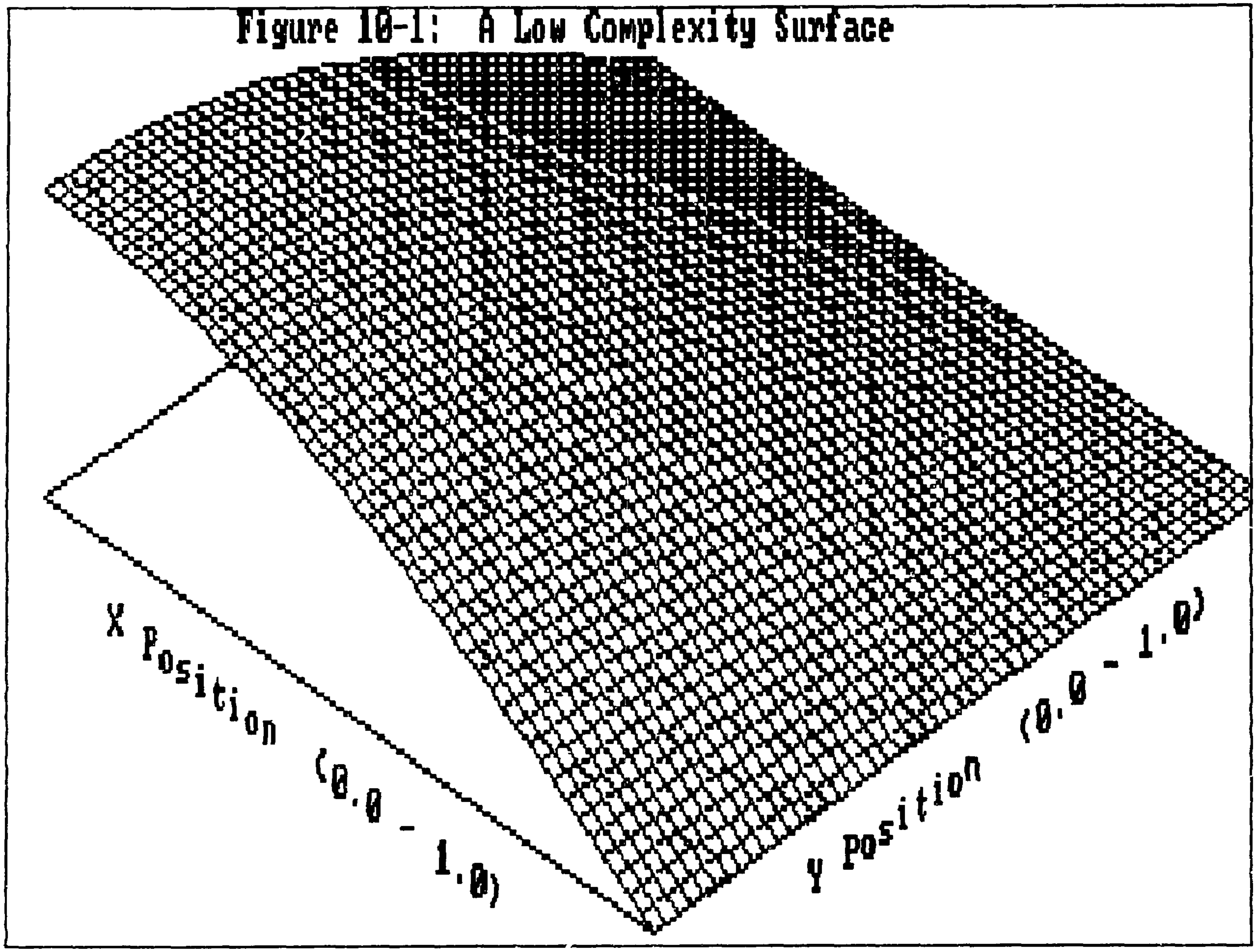

Figure 10-1. A Low Complexity Surface

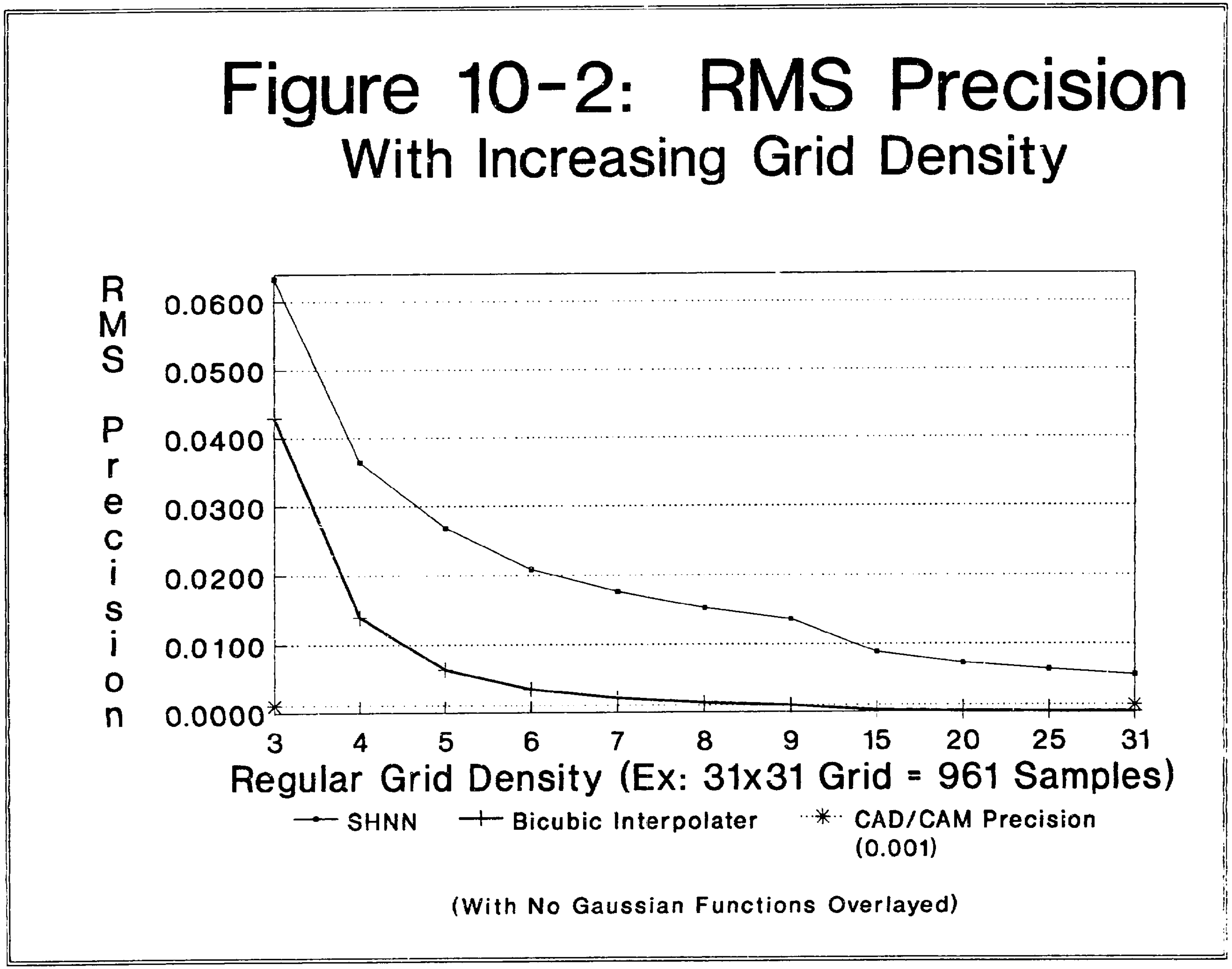

Figure 10-2. RMS Precision with Increasing Grid Density

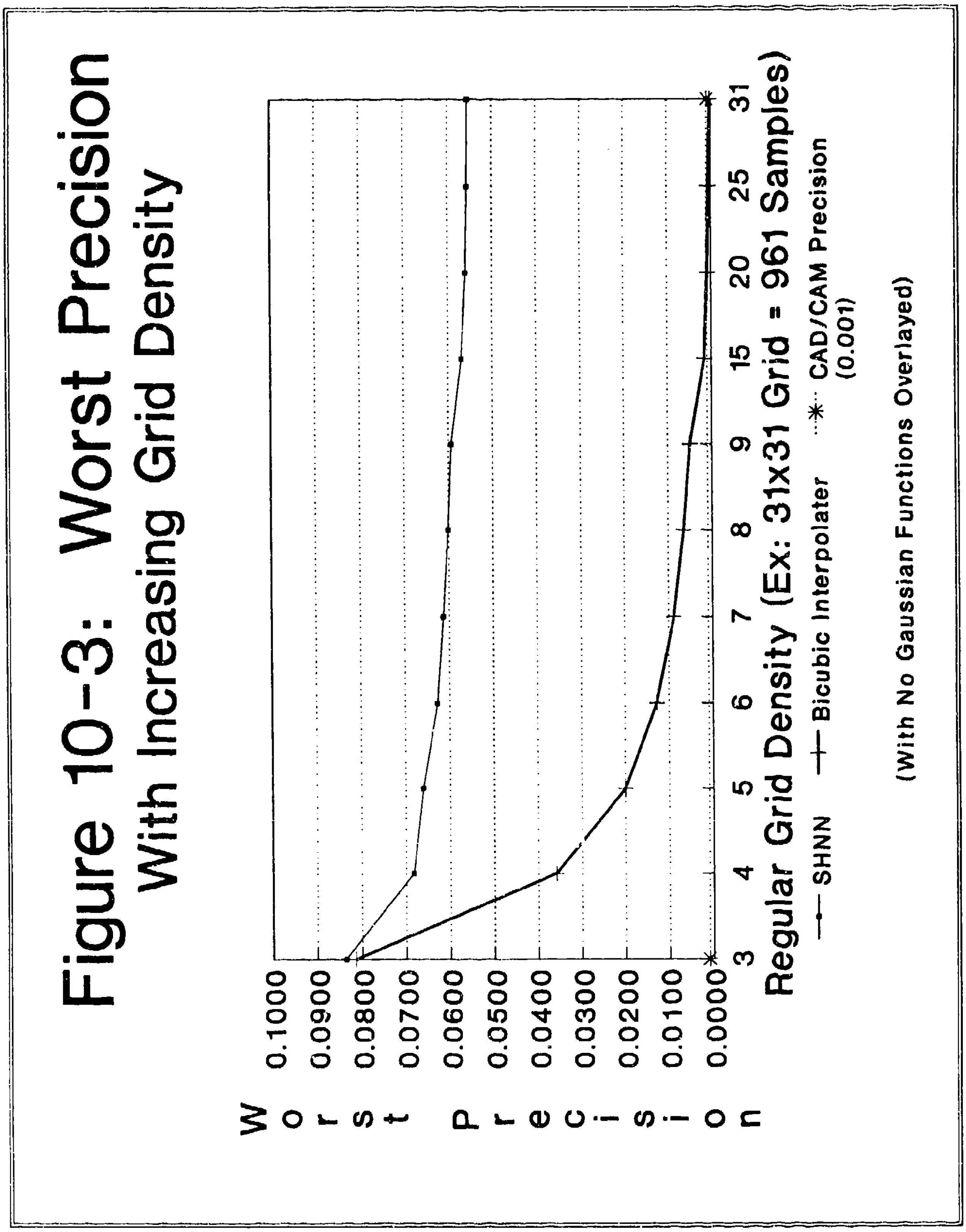

Figure 10-3. Worst Precision with Increasing Grid Density

Medium Complexity Surface

The surface of medium complexity used the same basic equation as the low complexity surface. However, a Gaussian bump was added on each of the four corners. The Gaussian bumps were generated using the following equation:

$$\textit{Surface Height} = \textit{Low Complexity Height} +$$

$$0.025\sum_{i=1}^{4} e^{\frac{-((g_{xi}-x)^2+(g_{yi}-y)^2)}{0.005}}$$

where: g_{xi} and g_{yi} are the x,y coordinates of the Gaussian centers
x and y are the x,y coordinates of a training point

A plot of this surface is shown in Figure 10-4 for a 50x50 grid. The same data ranges and grids as before were used.

Figure 10-5 compares the RMS precision of the SHNN to that of bicubic interpolation. Figure 10-6 compares the worst precision of the SHNN to that of bicubic interpolation.

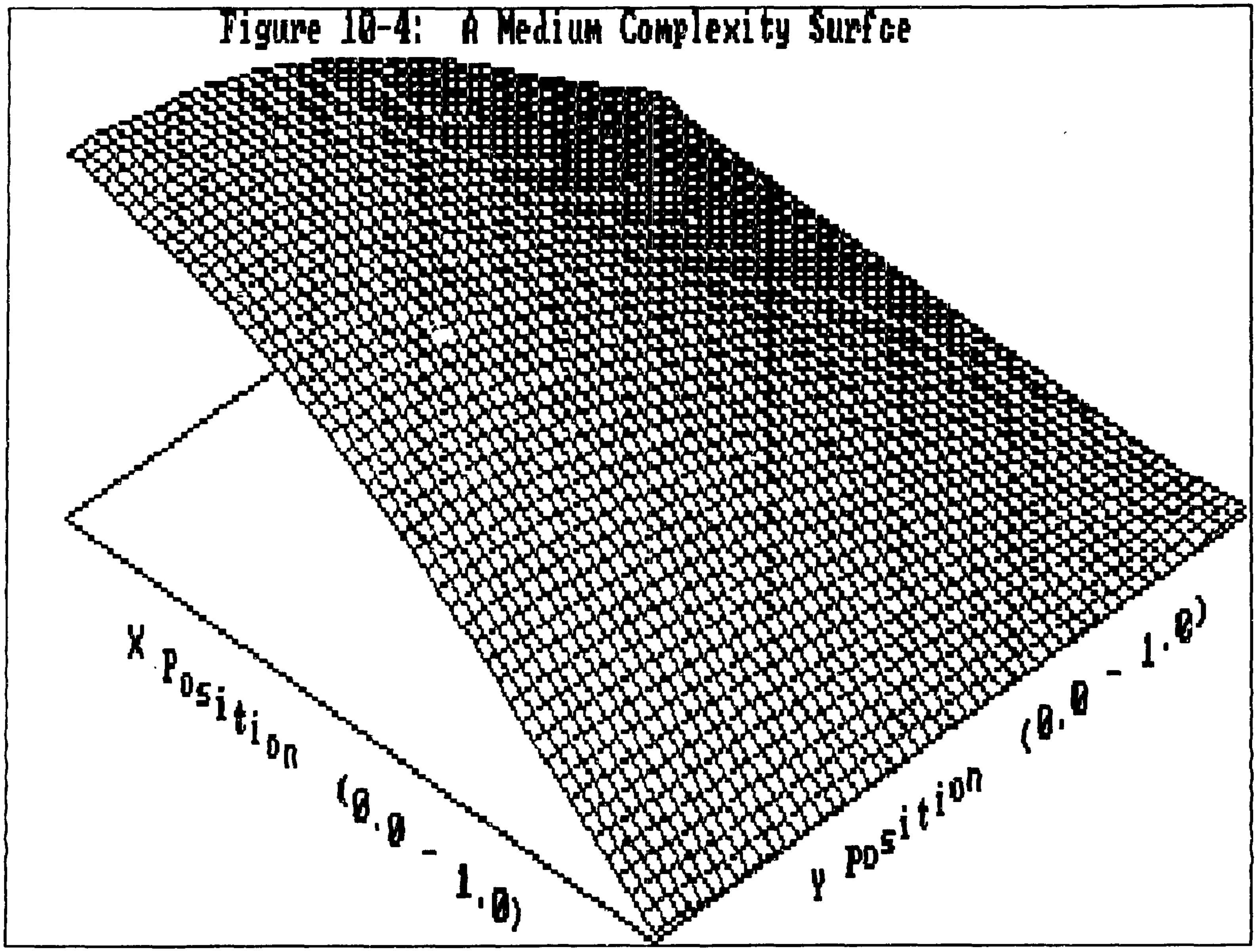

Figure 10-4. A Medium Complexity Surface

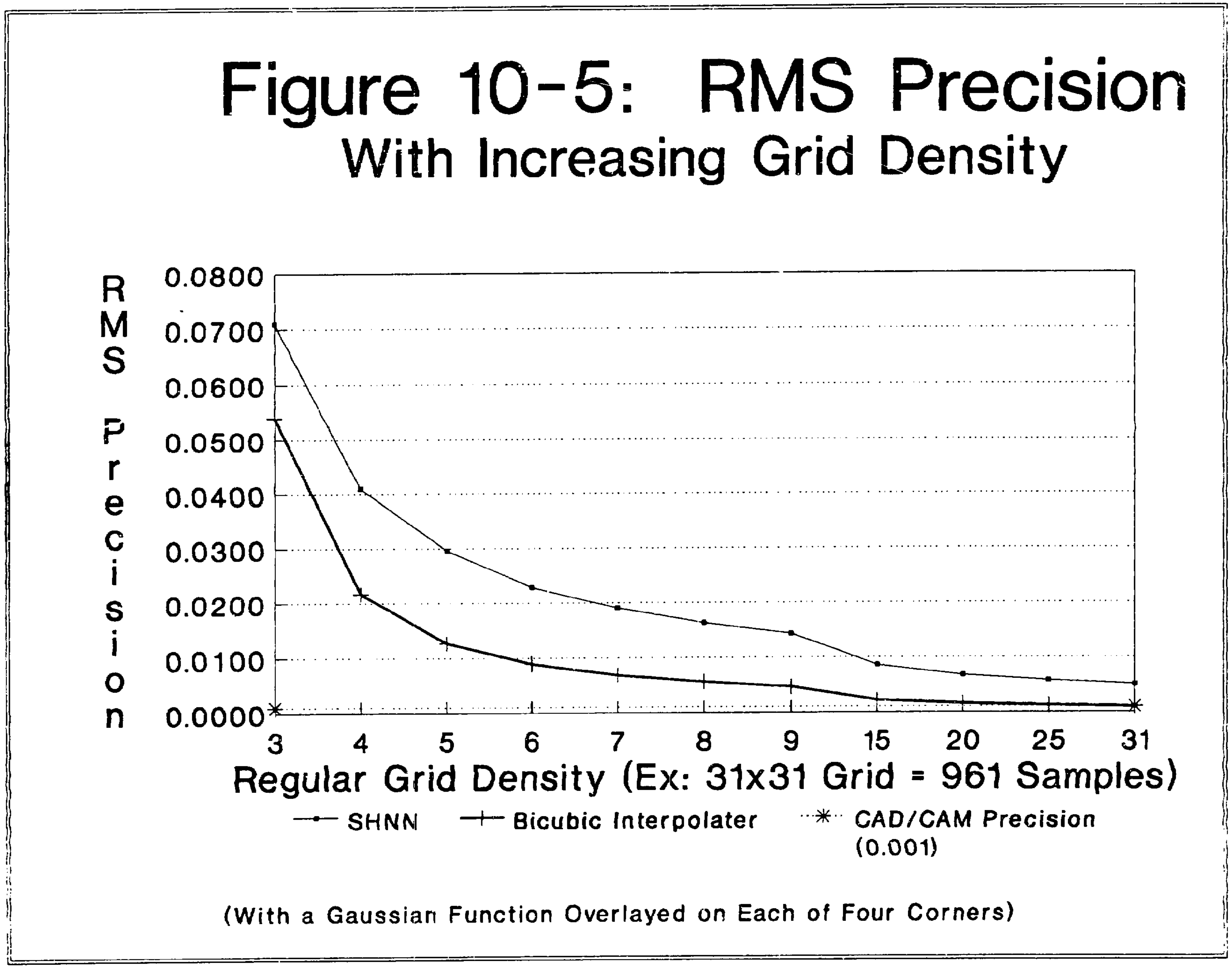

Figure 10-5. RMS Precision with Increasing Grid Density

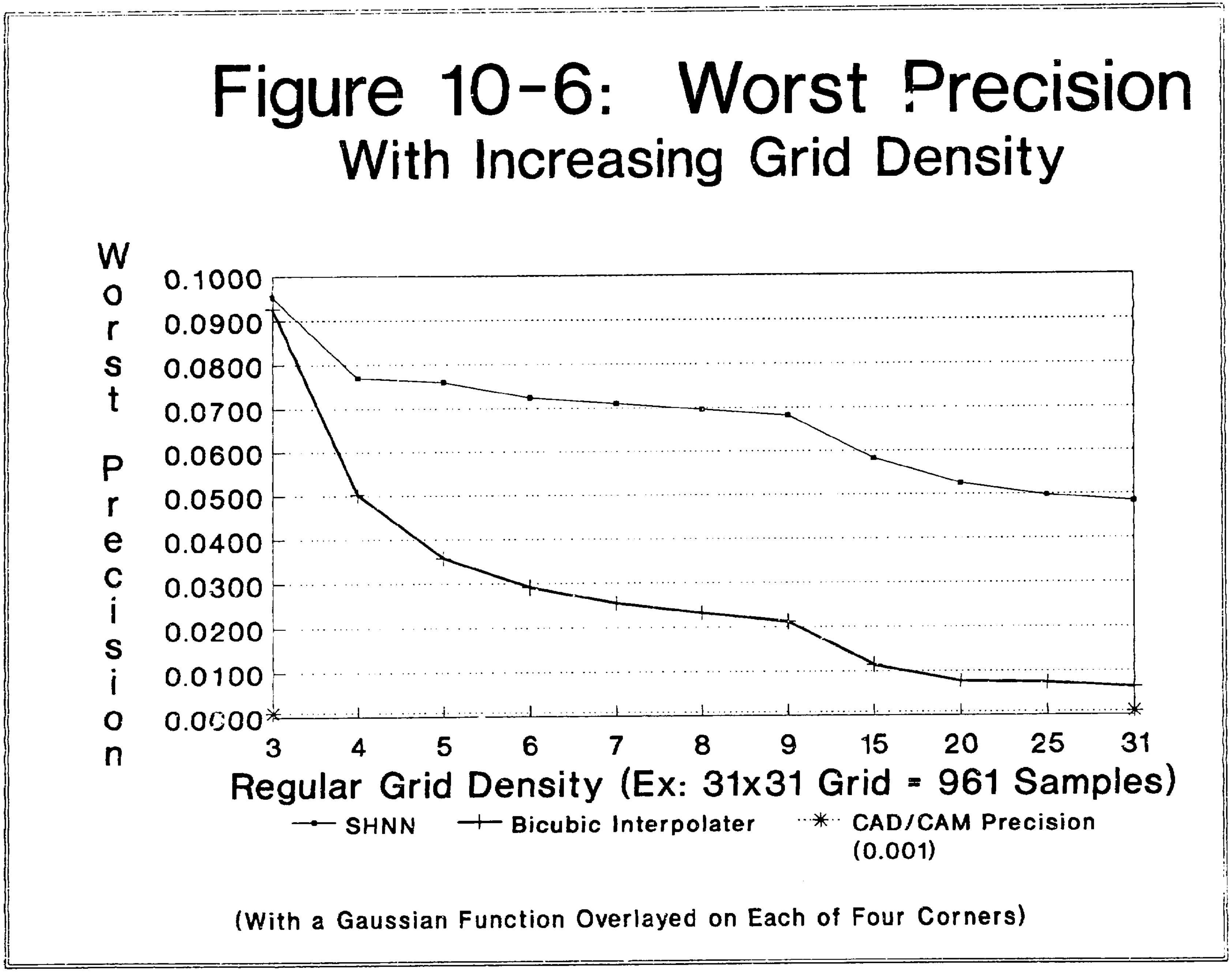

Figure 10-6. Worst Precision with Increasing Grid Density

High Complexity Surface

The surface of high complexity used the same basic equation as the low complexity surface. However, Gaussian bumps were added on an 8x8 grid which spanned the entire surface. The Gaussian bumps were generated using the following equation:

$$\textit{Surface Height} = \textit{Low Complexity Height} +$$

$$10.0\sum_{i=1}^{64} e^{\frac{-((g_{xi}-x)^2+(g_{yi}-y)^2)}{0.002}}$$

where: g_{xi} and g_{yi} are the x,y coordinates of the Gaussian centers
x and y are the x,y coordinates of a training point

A plot of this surface is shown in Figure 10-7 for a 50x50 grid. The same data ranges and grids as before were used.

Figure 10-8 compares the RMS precision of the SHNN to that of bicubic interpolation. Figure 10-9 compares the worst precision of the SHNN to that of bicubic interpolation.

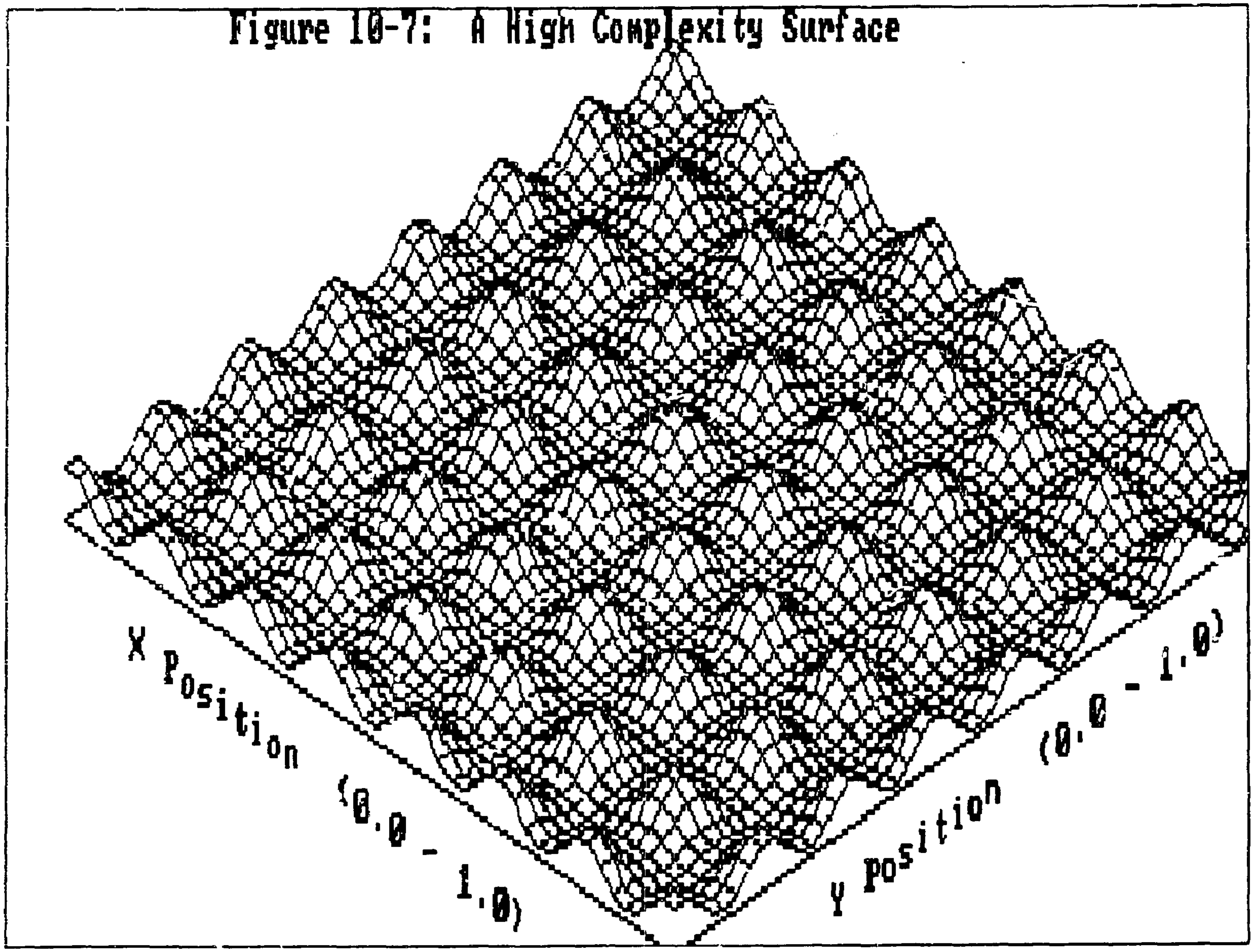

Figure 10-7. A High Complexity Surface

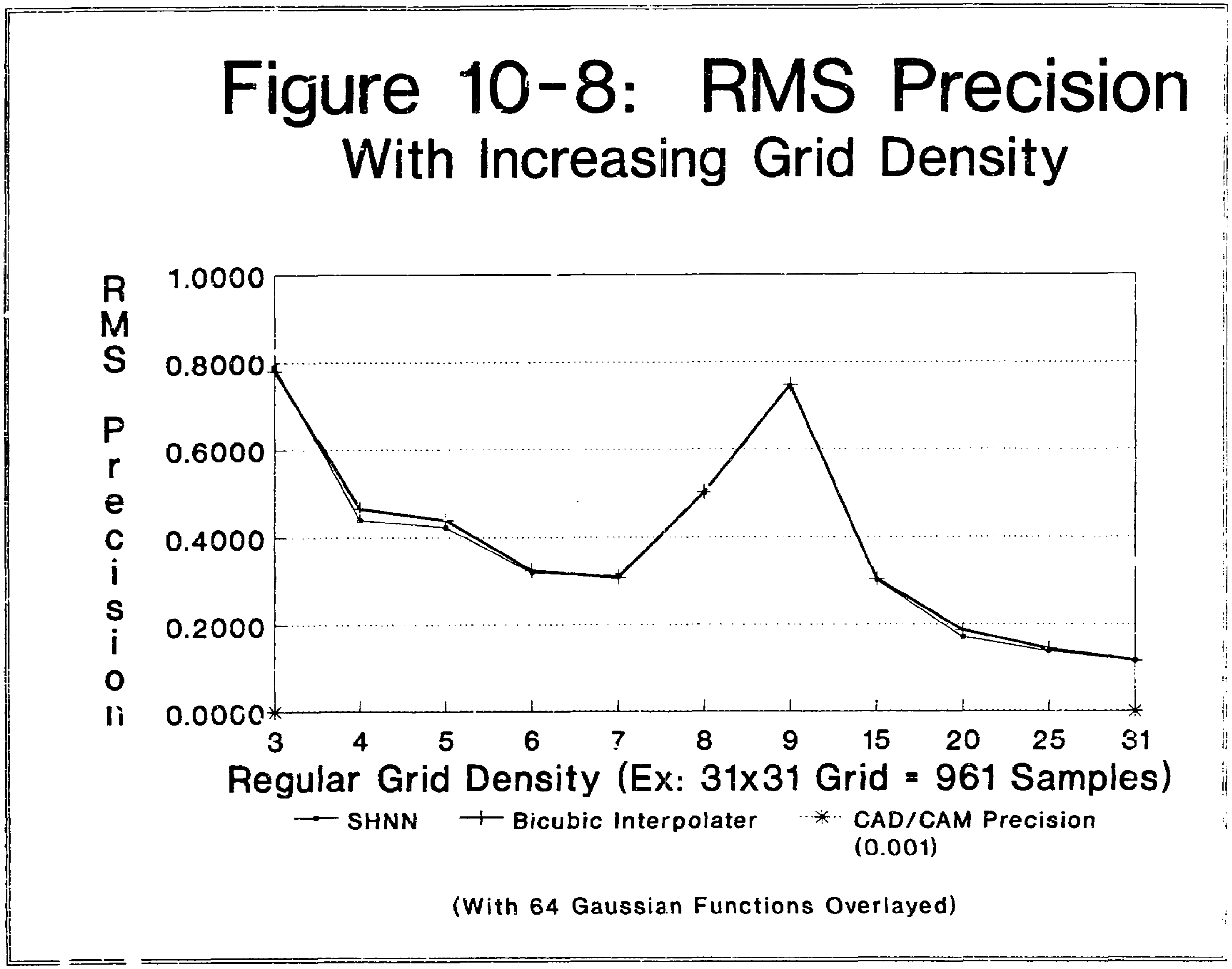

Figure 10-8. RMS Precision with Increasing Grid Density

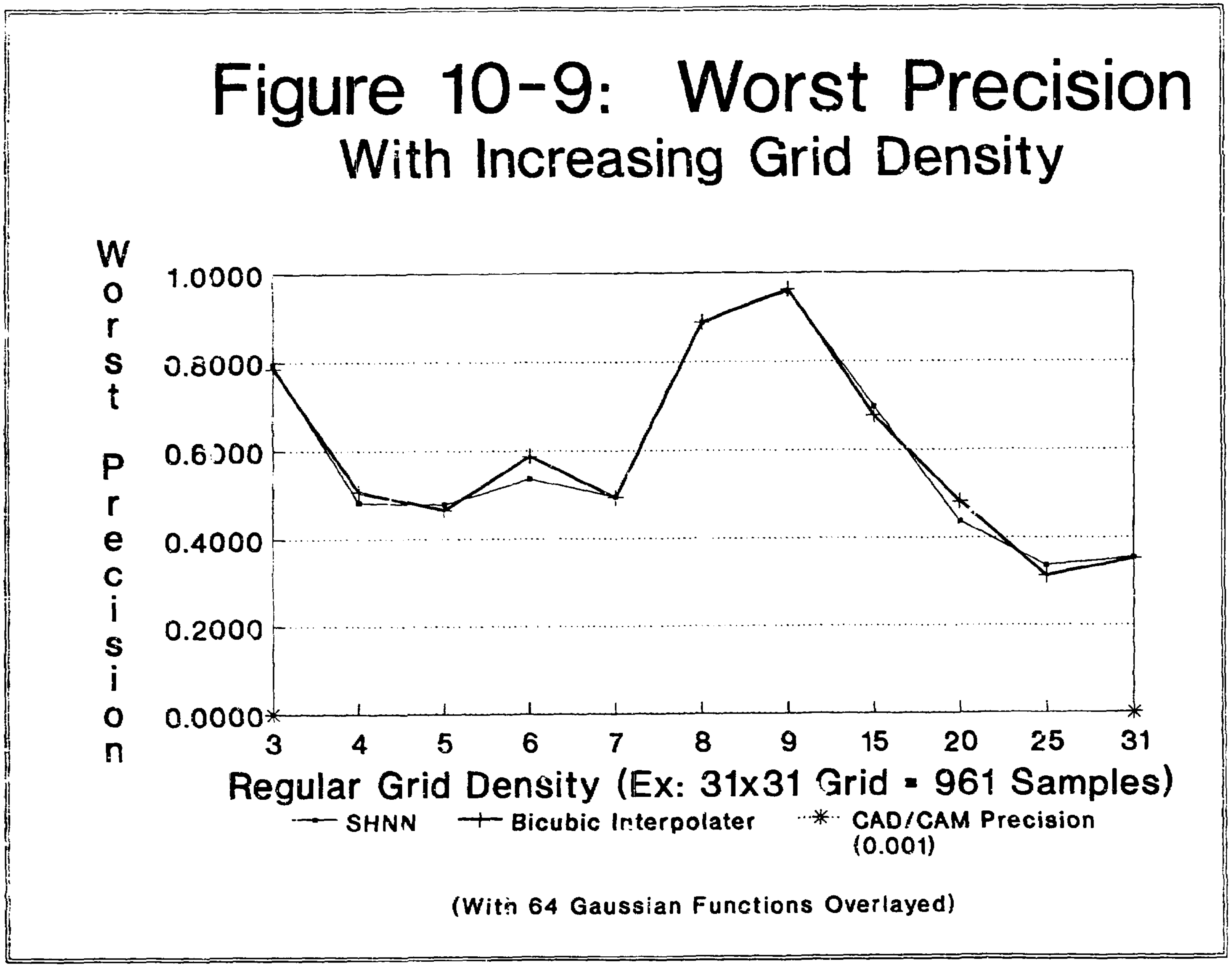

Figure 10-9. Worst Precision with Increasing Grid Density

11. Observations and Folklore

A key to the accurate functioning of neural networks based on Gaussian radial basis functions is appropriately setting the Gaussian variances. Specht's PNN has a single variance that is set by the user. The SHNN calculates a variance that is, generally, different for each training point. Thus, although the user could set desired widths, the SHNN can calculate optimal widths for best performance.

Only one pass through the training data is needed by the SHNN. This pass includes four places where iterations occur:

a) Finding the eigenvectors and eigenvalues
b) Solving the equations to fit the least squares plane to the training data
c) Adjusting the Gaussian variances to achieve columnar diagonal dominance
d) Solving the equations to determine the Gaussian coefficients

The longest iteration occurs when solving the equations that determine the Gaussian coefficients.

Solutions to systems of linear equations are generally of $O(N^3/c)$ computational complexity, where N is the number of equations and unknowns in a completely specified system. Singular Value Decomposition (SVD) is $O(N^3/3)$ and Lower-triangular:Upper-triangular Decomposition (LUD) is $O(N^3/15)$, according to our experience with those two methods.

When calculating columnar diagonal dominance, it is possible to use off-diagonal sums of less than 1. This possibility was investigated briefly. Our observation from a few experiments is that, while RMS precision is not greatly affected, there are opportunities to improve worst precision. The implication is that there may be ways to fine-tune the SHNN to minimize gross errors. Remember, all neural networks are heuristic by nature. They do not compute precise answers according to some algorithm. On the few surfaces we tried, an off-diagonal sum of 0.4 (rather than unity) worked best if there was any effect at all.

When using SVD for solving systems of equations, we have found that solutions to systems of equations are more readily achieved if the training vectors are first passed through a principal components transformation. In addition, we have found that the principal components transformation does not affect interpolation accuracy when regular grid sampling is used. However, even with regular grids, the principal components transformation speeds up the solution of systems of equations when SVD is used. In fact, SVD would not solve some systems we attempted without the principal components transformation.

It is best to use the C language precompiler called "Lint" on all code, especially that which is commercially procured. The version of Lint used must be oriented to your specific compiler. During this project, we were often surprised over how fragile C source can be due to compilable and executable coding errors such as coercion, promotion, and demotion. Lint will usually catch these types of errors. The C compilers typically will not catch such errors.

12. Conclusions Relative to 3D Solid Modeling

The authors have come to the following conclusions relative to 3D solid modeling.

For simple surfaces, traditional interpolation methods yield precision superior to that achievable by stretch and hammer's current implementation. (Bicubic interpolation was used as a base-line.) As the surface becomes more complex, the stretch and hammer neural network yields comparable precision.

For complex n-dimensional problems where the problem space cannot be sampled using an arbitrarily dense grid, traditional algorithms grow in complexity and execution time as n increases. Traditional methods grow in execution time as the number of training samples grows. The stretch and hammer neural network remains fixed in complexity (but not size) as n and the number of training samples grows. The SHNN remains fixed in execution time as n and the number of training samples grows if true neural hardware is used for implementation.

Traditional interpolation methods cannot extrapolate outside their training data without making certain assumptions and, in effect, adding training data. If the test data fall outside the training data, traditional methods cannot interpolate a surface height without the added training data. Thus, input data tolerances can cause values outside the realm of computability for traditional methods. The stretch and hammer neural network easily generalizes so that input data tolerances are accommodated.

Finally, stretch and hammer's parallel nature allows it to take advantage of off-shelf and future parallel computing hardware such as transputers and neural chips. Thus, there is strong reason to believe that the SHNN can execute in real time despite the problem complexity. Such may not be possible with traditional interpolation methods.

13. Two Other Applications

Filtering and classification are two other areas to which the SHNN may have useful application.

Filtering

An application of the stretch and hammer neural network is the synthesis of filters for an optical correlator used for pattern recognition. Figure 13-1 shows an optical correlator.

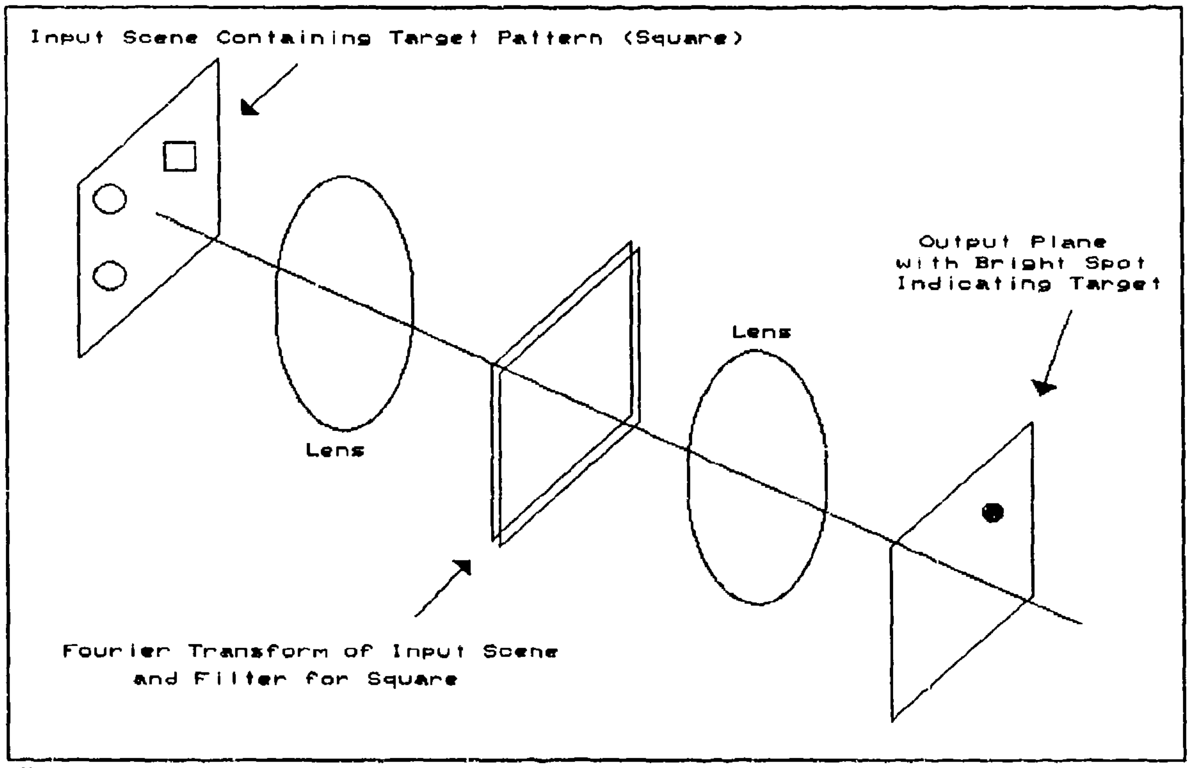
Input Scene Containing Target Pattern (Square)
Lens
Output Plane
with Bright Spot
Indicating Target
Lens
Fourier Transform of Input Scene
and Filter for Square

Figure 13-1: Optical Correlator

Coherent light from a pixelated input scene containing a target pattern passes through a simple lens one focal length away from the input. One focal length behind the lens, the Fourier transform (or spatial frequency spectrum) of the input scene is imaged. A pixelated transparency filter made from the Fourier transform of the target pattern to be recognized is also placed in this plane. Light passing through the filter is now the product of the Fourier transform of the input scene and the filter transparency. This light is then passed through another lens placed one focal length away. One focal length behind this second lens an image showing the correlation between the input scene and the target pattern is formed. Points of high correlation appear as bright spots on this output plane. Points on the input plane and the output plane can be matched in a one-to-one manner. Points of high correlation in the output plane (bright points) can be related to specific positions on the input plane that indicate the presence of the target pattern.

One major problem with this method is that the filter is made for a target pattern of specific orientation and scale (or size). If the target pattern in the input scene rotates or changes scale, then the original filter does not produce a strong correlation point in the output plane for the newly oriented target. It is possible that the SHNN can be used to synthesize new, more appropriate filters as the orientation and scale of the target pattern changes within the input scene. The input to the SHNN is data from the input scene which contains information about the orientation and scale of the target pattern. The output from the network is information about how to synthesize a new filter.

One specific implementation now being investigated at the University of Dayton Research Institute uses the following steps. First, it is assumed that the target pattern has been previously located in the input plane, but has since changed its orientation and scale. The values of the input scene in the region around the target location are sampled using a rectangular grid of points. The values from the sampling grid are used as the input values for the neural network. The network is trained by sampling the target pattern at orientations and scales for which the proper filter is known. The outputs of the network are the pixel values for the new filter which matches the sampled input from the reoriented and rescaled target pattern.

Classification

The SHNN may be employed to solve classification problems since classification is a sub-set of interpolation. In interpolation, the neural network determines the "height" of some hyper-surface at coordinates not included in the training set. In classification, the hyper-surface consists of "heights" that correspond to given classes. For instance, all coordinates (called features in classification) that are samples of the automobile class might have a height of 1. All features that are of none of the classes might be given a height of 0. Classes would have outputs valued sequentially using integers.

For classification the SHNN would be trained in the same way as for interpolation. The training set would include all the feature vectors and the hyper-surface "height" that corresponds to the appropriate class for each vector. Then, test vectors would be applied and an output would be calculated. The difference here is that the SHNN final output node would execute the output function int(output + 0.5) to truncate the decimal and ensure an integer class value.

A variation on this idea would have a post-processor layer that would take the raw SHNN output and determine if it represented a known class. Here, the classes would not have to be numbered sequentially (nor even be integers). Another advantage of this variation is that it would be possible to have an output value between two class values but not close enough to either to say it belonged to one class or the other. Thus, the opportunities to flag unknowns would increase.

The above approach to the use of the SHNN in classification makes little change to the original method. However, there are some weaknesses:

a) It is not possible, without transforming the coordinate axes, to collect "evidence" that multiple classes are indicated by the same feature vector.

b) A class with a low output value could be dominated by a, spatially, nearby class that has a very high output value.

c) Learning in real time would be difficult if not unlikely.

A solution is to create an SHNN with multiple outputs. Then connect the training sample nodes only to the output node(s) for the class(es) to which they apply. The expected output for all training vectors for all classes would then be fixed at some value. Class selections would be based on the highest-valued output node. If similarly high output values were produced by more than one node, then an ambiguity would result.

The least squares plane fits this case exactly. There is no difference between the surface value on the least squares plane at the training coordinates and the expected output at the training coordinates. Thus, the value of the initial Gaussian peaks would be zero. The resulting "hammered" surface would simply be the least squares plane itself, and there would be no surface variability that could be used to separate classes.

A solution to this dilemma is to "hammer" to the zero plane instead of the least squares plane. However, the difference between the value on the zero plane at the training coordinates and the expected output at the training coordinates is the expected output. Thus, the part of the training and the network used for "hammering" can be eliminated. The Gaussian coefficients would all have the same value. The Gaussian functions would generate the expected output at the training coordinates with the value decreasing as the distance from the training coordinates decreases. The expected output would be a value that is the same for all classes and training vectors. The Gaussian coefficients are equal to that expected value.

Columnar diagonal dominance might still be useful for determining the variances for each training point's Gaussian. It is possible that this technique could be used to ensure that the Gaussians have small overlap. During some brief trials with this idea, we found that the sum of the off diagonal column elements needs to be fairly small. We had success with sums of 0.05 and 0.1. All the training examples for all classes were placed into one large matrix for the diagonal dominance calculation.

If extra training sample nodes and class output nodes are added to the network, real time training could be implemented since there are no systems of equations to solve. The variance for any new training point could be something very narrow, the average of the two closest original training points, the average of all the original training points, or something else depending on the situation.

Specht's PNN is very similar to the Exponential Neural Network (ENN) outlined above. Specht's radial basis functions are Gaussian, but he varies the coefficient to ensure a volume of unity under each Gaussian. The PNN uses the same variance at each training point. The ENN subset of the SHNN maintains a Gaussian coefficient equal to the expected output and places Gaussian's with generally different variances at each training point.

14. Recommendations

Much work has been done, by various researchers, on basis function methods for programming massive arrays of parallel processors. This work should be consolidated since it offers much in the way of logistical supportability. Many other currently popular methods of programming these arrays are not logistically supportable.

A serious problem with the SHNN is that it does not train in real time for interpolation. Traditional interpolation methods train in real time and are more or as accurate as the SHNN. The SHNN has the advantage of constant execution time, given true neural hardware, in spite of increases in the dimensions and the numbers of training vectors. Traditional interpolation methods do not share this execution advantage. Efforts should be undertaken to develop a means of training the SHNN in real time for interpolation.

It would be interesting if neural network interpretations could be devised for traditional interpolation schemes. Such interpretations are not obvious but further study on the basic theories behind the traditional methods might reveal them. It is true that a pipeline interpretation is fairly easily made of traditional interpolation methods. However, the length of the pipe tends to grow as does the number of dimensions. Each joint on the pipe takes longer to execute as the number of training vectors grows. The advantage of the pipeline is that the joints execute in parallel. Upon receiving its input data, a joint can begin execution while earlier joints begin preparing the next round of input. With neural networks, the layers and the nodes in each layer execute in parallel. Like the pipeline joints, a layer receives its input data and begins executing. Previous layers then begin preparing the next round of input. Unlike the pipeline, the number of layers is fixed. As dimensions, training vectors, and classes are added, the number of nodes in various layers grows. The nodes in each layer always execute in parallel so there is never any change in the execution time of each layer if true neural hardware is used. Also unlike the pipeline, the neural network remains fixed in complexity. As the pipe grows in length with more joints added, a more sophisticated control method is needed. Neural networks simply change weight values from zero on inter-node connections to activate additional nodes.

Only the least squares plane and Gaussian radial basis functions were applied to the SHNN in this study. Explorations were briefly undertaken during this study to determine if other surfaces and functions would yield greater interpolation accuracy. These explorations should continue.

An effort to pin down the effect of off-diagonal sums of less than 1 should be made. An experimental approach would perform tests on surfaces of increasing complexity while applying to each surface an increasing density of sampling. For each surface, at each incremental density chosen, various off-diagonal sums, in a logical progression, could be tried.

On very complex surfaces, the SHNN has accuracy comparable to traditional interpolation schemes. However, we did not have the computing power to use the sampling density necessary to achieve CAD/CAM accuracy on the most complex surface. When it becomes possible, the sampling density should be significantly increased until CAD/CAM accuracy is reached. The goal is to see if the SHNN needs fewer samples than traditional methods need to reach CAD/CAM interpolation accuracy on very complex surfaces. To this end, the authors have submitted a grant proposal for time on the Ohio Super Computer Center's Cray. This proposal was approved after the research period that produced this report.

References

Donahue, Richard J. and Robert S. Turner, "CAD Modeling and Alternative Methods of Information Transfer for Rapid Prototyping Systems," Proc: International Conference on Rapid Prototyping, pp 221-235, Jun 91

Gustafson, Steve C.; Gordon R. Little; John S. Loomis; and Todd S. Puterbaugh, "Stretch and Hammer Algorithm for Data Representation and Generalization," Submitted to: Communications In Applied Numerical Methods, Mar 90

Hecht-Nielsen, Robert, Neurocomputing, Reading, MA: Addison-Wesley, p 329-330, 1990

Heller, Tim B., "Rapid Modeling - What is the Goal," Proc: International Conference on Rapid Prototyping, pp 246-248, Jun 91

Hertz, John; Anders Krogh; and Richard G. Palmer, Introduction to the Theory of Neural Computation, Redwood City, CA: Addison-Wesley, pp 204-206, 1991

Hotelling, H., "Analysis of a Complex of Statistical Variables Into Principal Components," Journal of Educational Psychology, vol 26, pp 139-142, 1935

Hull, Charles, "Rapid Prototyping in the 1990's," Proc: International Conference on Rapid Prototyping, pp 211-218, Jun 91

Kirschman, C.F.; A. Bagchi; C.C. Jara-Almonte; R.L. Dooley; and A.A. Ogale, "The Clemson Intelligent Design Editor for Stereo Lithography," Proc: International Conference on Rapid Prototyping, pp 240-245, Jun 91

Klimasauskas, Casimir; John Guiver; and Garrett Pelton, Neural Computing, Pittsburgh, PA: NeuralWare, Inc; 1989

Kruskal, Joseph B., "Multi-Dimensional Scaling and Other Methods for Discovering Structure," chapter in Statistical Methods for Digital Computers, Vol III, New York, NY: John Wiley and Sons, pp 296-339, 1977

Lippmann, Richard P., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, Apr 87

Lipschutz, Seymour, Linear Algebra, New York, NY: McGraw-Hill, pp 39-40, 1968

Maloney, P.S., "Use of Probabilistic Neural Networks for Emitter Correlation," Internal Report, Lockheed Missiles and Space Company, 1989

Nobel, Ben and James W. Daniel, Applied Linear Algebra, Englewood Cliffs, NJ: Prentice-Hall, pp 323-328, 1977

Press, William H.; Brian P. Flannery; Saul A. Teukolsky; and William T. Vetterling, Numerical Recipes in C - The Art of Scientific Computing, Cambridge, England: Cambridge University Press, 1989

Poggio, T. and F. Girosi, "Networks for Approximation and Learning," Proceedings of the IEEE, v78, pp 1481-1497, Sep 90

Raeth, Peter G., "Raising the Question of Backpropagation's Logistical Supportability," USAF Wright Laboratory internal report, Aug 91

Rogers, Steven; Matthew Kabrisky; Dennis Ruck; and Gregory Tarr, An Introduction to Biological and Artificial Neural Networks, Bellingham, WA: Optical Engineering Press, 1991

Rumelhart, David E. and James L. McClelland, Parallel Distributed Processing: Vol I, II, III, Cambridge, MA: MIT Press, 1986

Specht, D.F., "Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification," IEEE Transactions on Neural Networks, v1, pp 1111-1121, Mar 90

Specht, D.F., "Probabilistic Neural Networks for Classification, Mapping, or Associative Memory," Proc: International Conference on Neural Networks, 1988

Zahirniak, Daniel R., Characterization of Radar Signals Using Neural Networks, Dayton, OH: Air Force Institute of Technology, Thesis # AFIT/GE/ENG/90D-69, also appears in DTIC/NTIS, Dec 90

Author Biographies

Peter G. Raeth is an Air Force officer responsible for conducting, managing, and transitioning research programs that apply advanced information processing methods to problems in national defense. He received the Master of Science in Computer Engineering from The Air Force Institute of Technology in 1980. During the period of this present project he was on a fellowship with the University of Dayton Research Institute. His book on expert systems was published by the Institute of Electrical and Electronic Engineers (IEEE) Computer Society Press in 1990. He has published 13 papers and articles on neural network and expert system applications during the last 3 years. He is a member of Tau Beta Pi, Eta Kappa Nu, Omicron Delta Kappa, the IEEE Computer Society, and the Association of Old Crows.

Steven C. Gustafson is a Senior Research Physicist at the Research Institute of the University of Dayton. He received the Ph.D. in Physics from Duke University in 1974 and joined the University of Dayton in 1976. Since 1985 he has initiated, managed, and performed research on numerous government and industry contracts that have involved neural network applications and optical implementations. His other research activities emphasize electro-optics and have included the successful development of holographic optical processors for locating integrated circuit wafer defects, the design of adaptive optical processors for phased-array beam steering, and the development of intensity correlation techniques for passive optical device detection. Dr. Gustafson has also taught or cotaught several graduate engineering and workshop courses on neural networks and optics. He is author or coauthor of approximately 90 publicly available technical papers, proceedings, and reports, including approximately 30 in the neural network area since 1985. His professional memberships include IEEE, SPIE, OSA, SIAM, INNS, Sigma Xi, and Tau Beta Pi.

Gordon R. Little received the B.S., M.S. and Ph.D. degrees in Physics from the Ohio State University in 1966, 1970, and 1973 respectively. Following work at Systems Research Laboratories and Science Applications, Inc., he joined the University of Dayton in 1985 where he holds a joint appointment in the Research Institute and the University's graduate program in electro-optics. His research interests and activities include radiometry, ellipsometry, infrared systems, and optically implemented neural networks. He has published over 25 journal articles, conference papers, and technical reports in these areas.

Todd S. Puterbaugh is a graduate research assistant at the University of Dayton Research Institute and is currently working on his M.S. in Electro-Optics. He recieved his M.S. in Mathematics (1986) and B.S. in Physics (1984) from Miami University, Ohio. Prior to beginning his present research, he was a mathematician and computer scientist for Science Applications International Corporation (SAIC). He is a member of Sigma Xi, SIAM, and SPIE.

☆ U.S. Government Printing Office:1992—648-127/62.78